National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-55564-5

Canada

The University of Alberta

COMPARISON OF THINNING ALGORITHMS

by

Aji H. Wigena

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Fall, 1989

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Aji H. Wigena

TITLE OF THESIS: Comparison of Thinning Algorithms

DEGREE FOR WHICH THIS THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1989

Permission is hereby granted to The University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) .....

Permanent Address:
Kampung Duri Pasar No.35
Jakarta 11270, DKI Jakarta Raya
Indonesia

Dated, July 4th, 1989.

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Comparison of Thinning Algorithms** submitted by **Aji H. Wigena** in partial fulfillment of the requirements for the degree of **Master of Science.**

Xiaobo Li

Supervisor

W. A. Davis

Date July 4th. 1989

ABSTRACT

This thesis defines several analytical criteria, reviews several well-known thinning algorithms, investigates a lookup table reformulation, and then proposes a new thinning algorithm. An ideal skeleton should satisfy the criteria, i.e. connectivity, thinness, center position, shape adaptivity, stability, and noisy branches. These criteria have been incorporated in the algorithms by certain rules and operations.

Two families of thinning algorithms are the algorithms with subiterative operations and those with contour-tracing operations. The second algorithms are faster than the first ones, and the algorithms with two subiterations are faster than those with four subiterations.

Instead of direct computations, deletion rules can be implemented as a table lookup, which is usually faster but requires more storage. The table is reformulated so that one small table can be used for the methods and requires small storage. This table reformulation is flexible and can be used as an experimental tool. However, the algorithms using the reformulated tables are slower.

The new thinning algorithm, which is quite different from the existing methods, generates better skeletons but requires more computation time.

Acknowledgements

I would like to thank my supervisor, Dr. Xiaobo Li, for his valuable advice, guidance and support throughout this research and the production of this thesis. Thanks are due to the thesis examining committee members, Dr. Wayne Davis, Dr. Joe Culberson and Dr. Walter Bischof, for their time and suggestions. Thanks also to the committee chairman, Dr. Wlodek Dobosiewicz, for his time during the oral examination; to Ms. Carol Smith for helping in preparing this thesis; and to Ms. Susan B. El-Nahhas, for her proofreading.

I am also indebted to World University Service of Canada (WUSC) for administering the scholarship programme. Funding assistance was provided by the World Bank through the Ministry of Education and Culture, Directorate General of Higher Education in Indonesia.

I wish to express my appreciation to my parents, my wife, Evi Hartati, and my children, Nui, Nea and Nanda, for their patience, encouragement and support.

Table of Contents

Chapter	Page
Chapter 1: Introduction	1
1.1. Criteria, Algorithms, and Skeletons	2
1.2. Table Lookup Approach	2
1.3. A New Thinning Method	3
1.4. Thesis Objective and Organization	3
Chapter 2: Existing Thinning Algorithms	4
2.1. Definitions and Notations	4
2.2. Subiterative Algorithms	11
2.2.1. Chen-Hsu's Algorithm (CH)	11
2.2.2. Zhang-Suen's Algorithm (ZS)	14
2.2.3. Deutsch's Algorithm (DE)	15
2.2.4. Naccache-Shinghal's Algorithm (NS)	16
2.2.5. Tamura's Algorithms (T1 and T2)	17
2.3. Algorithms with Contour-Tracing	18
2.3.1. Govindan-Shivaprasad's Algorithms (G1 and G2)	18
2.3.2. Pavlidis' Algorithm (PA)	20
2.4. Criteria	21
2.4.1. Connectivity	21
2.4.2. Thinness	23
2.4.3. Position	23
2.4.4. Stability	23
2.4.5. Shape-Adaptive	24
2.4.6. Noisy Branch	25
2.4.7. Sensitivity	26
2.5. Comparisons	31
2.5.1. Skeletons and Criteria Satisfied	31
2.5.2. Time Complexity	33
Chapter 3: A Table Reformulation	35
3.1. Lookup Tables	35
3.2. Reformulation	38
3.3. Comparison	45
3.3.1. Storage Requirement for Table Entries	45
3.3.2. Time Requirement	46
3.3.3. Skeletons	47

3.4. A New Set of Sequence Patterns 47
3.5. Summary 52
Chapter 4: A Proposed Thinning Algorithm 53
4.1. Windows and Patterns 53
4.1.1. Pcodes 53
4.1.2. Windows 54
4.1.3. Window Patterns 55
4.2. New Thinning Algorithm 57
4.2.1. Satisfying Criteria 58
4.2.2. Sensitivity 59
4.2.3. Time Complexity 60
4.3. Results and Comparisons 61
4.3.1. Skeletons and Criteria Satisfied 61
4.3.2. Time and Storage Requirement 65
4.4. Summary 66
Chapter 5: Conclusion and Future Work 67
5.1. Conclusions 67
5.2. Further Work 68
References 69
A1: Rule Windows 71
A2: Results of Existing Thinning Algorithms 77
A3: Results of Experiment on Sensitivity 104
A4: Results of Reformed CH, T2 and G2 112

List of Tables

Table		Page
2.1	Operations and Numbers of Rule Windows	21
2.2	Computation Time for Evaluating CH Rules	34
2.3	Time Requirement of Algorithms	34
3.1	Deletion Rules for S-E Subiteration	36
3.2	Deletion Rules for N-W Subiteration	37
3.3	Computation Time for Finding an Index	40
3.4	Sequence Patterns and Decision Values 'D'	43
3.5	Numbers of Table Entries 'D'	44
3.6	Storage Requirement for Table Entries	46
3.7	Time Requirement of Algorithms CH, T2 and G2	46
4.1	Components of Processing Time	61
4.2	Time and Storage Requirement of New Algorithm and G2	66

List of Figures

Figure	Page
2.1 Binary Image of Object 'C'	5
2.2 8-neighbors of Pixel *p*	6
2.3 Contour of Object 'C'	6
2.4 Freeman's Codes	7
2.5 Incode and Outcode	7
2.6 1-to-0 Pixels Transitions	8
2.7 Endpoint Pixel *p*	9
2.8 Skeletal Pixels *p*	10
2.9 Rule Windows of CH (N-W subiteration)	13
2.10 Rule Windows of CH (S-E subiteration)	13
2.11 Rule Windows of ZS (S-E subiteration)	14
2.12 Rule Windows of DE (N-E subiteration)	15
2.13 Rule Windows of NS (E-W subiteration)	16
2.14 Rule Windows of T1 (S subiteration)	17
2.15 Rule Windows of T2 (S subiteration)	17
2.16 Rule Windows of G1	19
2.17 Corner Windows	19
2.18 Stairlike Skeletons	22
2.19 Rule Windows for 4-connectivity	22
2.20 Rule Windows for 8-connectivity	22
2.21 Endpoint Shrinkage Skeletons	24
2.22 Non-shape Adaptive Skeleton	25
2.23 Noisy-branch and Non Noisy-branch Skeletons	26
2.24 Skeletons of 'B' with Noise	27
2.25 Skeletons of 'T' with Noise	28
3.1 Values of 8-neighbors	36
3.2 60 Different Neighbor Windows in CH	38
3.3 15 Different Neighbor Windows in CH	39
3.4 Possible Windows for 90- and T-Corners	48
3.5 Skeletons produced by New Set	49
4.1 Pcodes	54
4.2 Windows of c-pixel	54
4.3 Window Patterns	55
4.4 Window Patterns for Pcode 13	56
4.5 Noise	59

4.6 Skeletons generated by the New Algorithm 62
A1.1 Rule Windows of ZS (N-W subiteration) 71
A1.2 Rule Windows of DE (S-W subiteration) 72
A1.3 Rule Windows of NS (N-S subiteration) 73
A1.4 Rule Windows of T1 (W subiteration) 73
A1.5 Rule Windows of T1 (E subiteration) 74
A1.6 Rule Windows of T1 (N subiteration) 74
A1.7 Rule Windows of T2 (W subiteration) 74
A1.8 Rule Windows of T2 (E subiteration) 75
A1.9 Rule Windows of T2 (N subiteration) 75
A1.10 Rule Windows of PA 76
A2.1 Skeletons of 'A' 78
A2.2 Skeletons of 'B' 79
A2.3 Skeletons of 'C' 80
A2.4 Skeletons of 'D' 81
A2.5 Skeletons of 'E' 82
A2.6 Skeletons of 'F' 83
A2.7 Skeletons of 'G' 84
A2.8 Skeletons of 'H' 85
A2.9 Skeletons of 'I' 86
A2.10 Skeletons of 'J' 87
A2.11 Skeletons of 'K' 88
A2.12 Skeletons of 'L' 89
A2.13 Skeletons of 'M' 90
A2.14 Skeletons of 'N' 91
A2.15 Skeletons of 'O' 92
A2.16 Skeletons of 'P' 93
A2.17 Skeletons of 'Q' 94
A2.18 Skeletons of 'R' 95
A2.19 Skeletons of 'S' 96
A2.20 Skeletons of 'T' 97
A2.21 Skeletons of 'U' 98
A2.22 Skeletons of 'V' 99
A2.23 Skeletons of 'W' 100
A2.24 Skeletons of 'X' 101
A2.25 Skeletons of 'Y' 102
A2.26 Skeletons of 'Z' 103
A3.1 Skeletons produced by CH 105
A3.2 Skeletons produced by ZS 106

A3.3 Skeletons produced by DE 107
A3.4 Skeletons produced by T2 108
A3.5 Skeletons produced by NS 109
A3.6 Skeletons produced by G2 110
A3.7 Skeletons produced by PA 111
A4.1 Skeletons produced by Reformed CH 113
A4.2 Skeletons produced by Reformed T2 116
A4.3 Skeletons produced by Reformed G2 119

Chapter 1

Introduction

In pattern recognition, it is necessary to recognize a line like object in a binary image. Usually, the thickness variation of the object does not contribute to the recognition process. Thinning, which can extract a uniform thickness skeleton of the object, is preprocessing to simplify further processing, such as character recognition. Characters should be thinned before the recognition process. Skeletons may be represented by chain codes to reduce data storage and transmission time. Thinning has also been used in fingerprint identification, chromosome analysis, soil-cracking pattern, and facsimile transmission [ChS86, DaP81, GoS87]. Many thinning algorithms have been developed and improved for better skeletons and less computation time.

All thinning algorithms are iterative. In each iteration, object pixels are examined against a set of deletion rules. The result of an iteration is the deletion of the outermost layer of pixels. Every iteration can be implemented either by several subiterations or by a contour-tracing. The contour-tracing approach is faster. The deletion rules may be implemented either as a direct computation or as a table lookup [ChH88].

The study of thinning algorithms is an active research area. Recently, Kwok [Kwo88] proposed a contour generating serial algorithm and compared it to four existing algorithms. Hall [Hal89] improved a parallel algorithm and compared the modified algorithm to a subiterative algorithm in terms of the number of iterations. Connectivity preservation is analyzed. Guo and Hall [GuH89] modified a subiterative algorithm and proposed an algorithm using a 2-subfield method. Plamondon and Suen [PlS88] reported three types of experiments to compare 11 thinning algorithms and defined reference skeletons resulted from human thinning. The skeletons generated by the algorithms are compared based on the reference models.

This thesis attempts to define several analytical criteria and review several well-known thinning algorithms. A general table reformulation of these algorithms is described as an experimental tool. Then, a new thinning algorithm is proposed.

1.1. Criteria, Algorithms, and Skeletons

The criteria for thinning output [GoS87, Hil69, StR71] are:

1. connectivity,
2. thinness,
3. position,
4. stability,
5. shape adaptivity,
6. noisy branches.

These criteria will be discussed in detail along with existing thinning algorithms in Chapter 2. According to the criteria, a skeleton should be connected, have a unit thickness, be in the center of the original object, have no endpoint shrinkage, be shape adaptive, and have no noisy branches.

Basically, thinning is a local process. However, in order to obtain a "good" skeleton, global shape information of the object must be taken into account. Many thinning algorithms have attempted to incorporate the global properties, but different algorithms generate different skeletons [ChS86, DaP81, ZhS84], and no single algorithm is completely satisfactory in every case.

1.2. Table Lookup Approach

Deletion rules in many algorithms can be implemented by table lookup which is usually faster but requires more storage. We modified the table for several existing algorithms so that much less space is needed. A table contains only cases of object pixels that should be deleted, which are about 4% of all possible cases. However, this small table requires a longer search time than the original large table. The modified table lookup approach also provides flexibility as the table can be easily modified for the algorithms or for various versions of one algorithm. We present the

table lookup approach and a table reformulation as a powerful experimental tool.

1.3. A New Thinning Method

An important goal of the thinning process is to preserve the global properties of the object. We propose a new thinning algorithm with contour-tracing that applies some square-window patterns to every contour pixel to maintain the topology of the object contour. This method generates better skeletons, but requires more computation time than existing methods.

1.4. Thesis Objective and Organization

This thesis reviews several well-known thinning algorithms, investigates a table reformulation of these methods, and proposes a new thinning algorithm.

Algorithm comparison based on time and storage requirement is straightforward. The comparison of skeletons is difficult. Usually, skeletons are visually compared based on: connectivity, center position, endpoint shrinkage and shape distortion. This thesis compares the performance of different thinning algorithms through extensive experiments. The results are reported in Chapter 2.

Chapter 3 presents the table reformulation method. The storage and time requirements of the algorithms with the reformulated table are compared to those of the original algorithms.

A new thinning algorithm is proposed in Chapter 4. The experimental results of this algorithm and a comparison with other algorithms are reported. A brief conclusion is given in Chapter 5.

Chapter 2

Existing Thinning Algorithms

Various thinning algorithms have been proposed. Different rules and operations result in different skeletons of the same object [DaP81]. In this chapter, the behavior of several thinning algorithms is reviewed and compared. The comparison is based on the skeletons, whether or not certain criteria are satisfied. The criteria and sensitivity are also discussed.

Section 2.1 presents some definitions and notation used through this thesis. Sections 2.2 and 2.3 discuss algorithms with subiterative and contour-tracing operations, respectively. Criteria and sensitivity are described in Section 2.4 and the algorithms are compared in Section 2.5.

2.1. Definitions and Notations

Before discussing the details of the algorithms, some basic terminologies are defined.

Binary Images

In a rectangular grid, an image may be considered to be a matrix whose rows and columns identify pixels in the image and the corresponding matrix element, $f(i,j)$, identifies the grey level at the pixel (i,j). A *binary* image refers to the matrix where each element is either 1 (black) or 0 (white). Usually, the object consists of those pixels that have value 1 (*1-pixels*). Fig. 2.1. shows a binary image with object character 'C'.

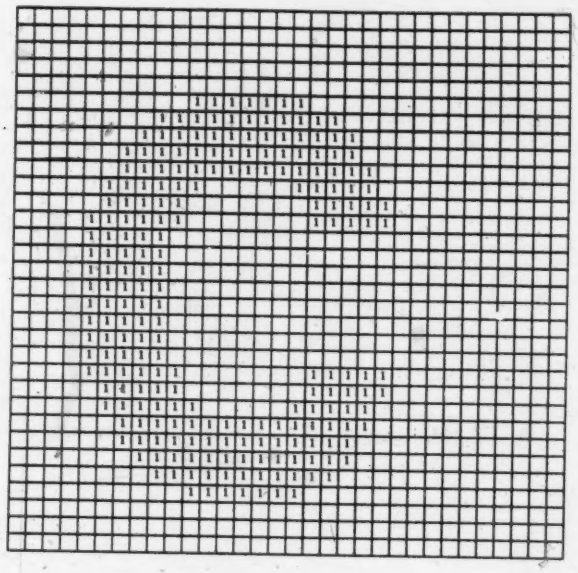

Figure 2.1 Binary Image of Object 'C'

Neighborhood

In a 3×3 window, there are two kinds of neighbors of a pixel, i.e. 8-neighbors and 4-neighbors. The *8-neighbors* of a pixel are the elements of $N_8 = \{n_0, n_1, \ldots, n_7\}$, where the pixels n_i, $0 \le i \le 7$, are shown in Fig. 2.2. The *4-neighbors* of the pixel are the elements of $N_4 = \{n_0, n_2, n_4, n_6\}$. The *neighbor path* of a pixel p is a clockwise 8-pixel sequence of 8-neighbors of p, starting from any 8-neighbor pixel of p; for example, the sequence of $n_2 n_3 n_4 n_5 n_6 n_7 n_0 n_1$.

n_7	n_0	n_1
n_6	p	n_2
n_5	n_4	n_3

Figure 2.2 8-neighbors of Pixel p

Contour and Chain Codes

The *contour* of an object O is defined as the set of pixels in O which have at least one 4-neighbor not in O. The contour of object 'C' (Fig. 2.1) is the set of pixels marked by 'c' *(c-pixels)*, as shown in Fig. 2.3.

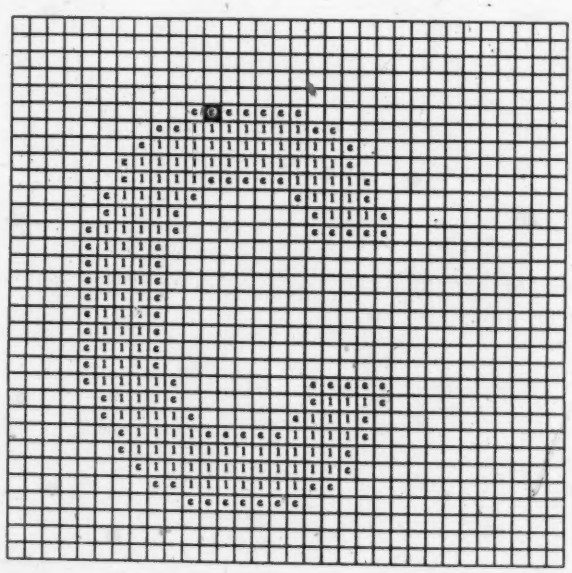

Figure 2.3 Contour of Object 'C'

Fig. 2.4 shows Freeman's codes [GoS87] to denote relative locations of neighbor pixels with respect to p, or directions from the pixel p to its neighbor pixels. Based on the codes and clockwise contour tracing, the *chain code* of the contour of object 'C', starting from the pixel *(5,11)*, is 22222232343 ... 01121. Every c-pixel has a *pcode*, which is a pair of codes, *incode* and *outcode*. Suppose three c-pixels q, r and s in Fig. 2.5 are in sequence in a chain code. The incode of r is the direction from q to r or the code of r (as a neighbor of q), and the outcode of r is the direction from r to s or the code of s (as a neighbor of r). For instance, q is pixel (5,15), r is pixel (5,16) with the code 2, and s is pixel (6,17) with the code 3, so the pixel (5,16) has incode 2 and outcode 3. The pcode of r is 23.

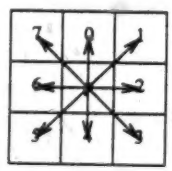

Figure 2.4 Freeman's Codes

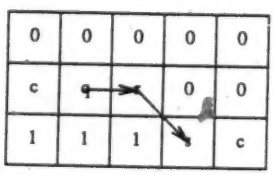

Figure 2.5 Incode and Outcode

Connectivity and Crossing Number

In dealing with objects on a rectangular grid, there are two kinds of connectedness, i.e. 8-

connected and 4-connected, so that the result of the thinning process preserves either *8-connectivity* or *4-connectivity*. Two pixels, p_0 and p_m, are called *8-connected* if there exists a path of pixels $p_0 p_1, \ldots p_m$ such that p_{i+1} is an 8-neighbor of p_i, $0 \le i < m$. Two pixels, p_0 and p_m, are called *4-connected* if there exists a path of pixels $p_0 p_1, \ldots p_m$ such that p_{i+1} is a 4-neighbor of p_i, $0 \le i < m$.

Connectivity is related to the *crossing number (CSN)* which is defined as, [GoS87, Rut66]

$$CSN = \tfrac{1}{2} \sum_{i=0}^{7} |n_i - n_{i+1}| \qquad (2.1)$$

where:

$$n_{i+1} = n_0 \text{ if } i+1 = 8$$

The crossing number indicates the number of 1-to-0 pixel transitions along the neighbor path of pixel p. Generally, $CSN=1$ results in 4-connected skeletons, and $CSN>1$ results in 8-connected or unconnected skeletons. For example, consider the three windows in Fig. 2.6. The first window shows only one 1-to-0 pixel transition, so $CSN=1$. If the pixel p is deleted, the result is 4-connected. The second window shows three transitions which will result in 8-connectedness, while the third window shows two transitions which will give an unconnected result, and CSNs of the two windows are 3 and 2, respectively. Equation (2.1) will also give the same CSN values.

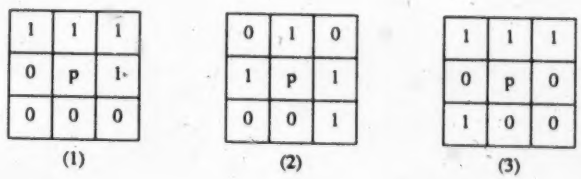

Figure 2.6 1-to-0 Pixels Transitions

Black Pixels

The *black pixel (BP)* of a pixel p is defined as the number of 1-pixels in the neighborhood of p,

$$BP(p) = \sum_{i=0}^{7} n_i \tag{2.2}$$

For example, BP=4 for the pixels p in the three windows in Fig. 2.6.

Endpoint Pixels

A pixel p is called an *endpoint* if it has a value 1 and has exactly one neighbor that has a value 1, i.e. BP=1. Fig. 2.7 shows an endpoint.

0	0	0
0	p	0
0	0	1

Figure 2.7 Endpoint Pixel p

Skeletal Pixels

A pixel p in a connected set of pixels is called *skeletal* if its removal from the set affects connectivity [GoS87, Pav81]. The pixels p in Fig. 2.8 are skeletal pixels if at least one a and one b have values 1.

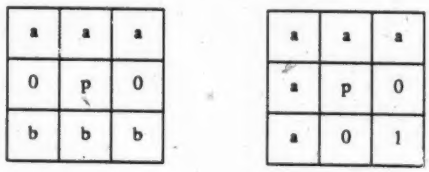

Figure 2.8 Skeletal Pixels p

Operations

The thinning process is started from the outermost layer of pixels of an object until skeletal pixels are found. The thinning process can be implemented iteratively either by a *subiterative* or by a *contour-tracing* operation. The subiterative operation is based on the directions of north (N), west (W), south (S), and east (E). Every iteration consists of 2 or 4 subiterations which are sequentially implemented. For example, a 2-subiterative operation may consist of NW and SE subiterations, and a 4-subiterative operation may consist of N, W, S, and E subiterations. In each subiteration, pixels are examined and deleted simultaneously, where processing pixels at the zth iteration depends on the pixel values at the $(z-1)$th iteration. In a contour-tracing operation, the contour pixels are examined and deleted sequentially, i.e., one pixel at a time. Processing a pixel at the zth iteration depends on the pixel values at the same iteration.

There are two families of algorithms based on the two kinds of operations, i.e. algorithms with subiterative and contour-tracing, which are discussed in the next sections.

2.2. Subiterative Algorithms

The general form of a subiterative algorithm is as follows.

```
do {  /* iteratively */
    for every subiteration, /* 2 or 4 subiterations */
        do {
            for every 1-pixel,
                if the neighbors of the pixel satisfy the deletion rules in the
                corresponding subiteration,
                    then mark the pixel by 'd';
        }
        do {
            for every pixel marked 'd',
                change the value of the pixel from 1 to 0;  /* delete */
        }
} until (no more pixel deletion);
```

In each subiteration, an image is scanned from left to right and top to bottom. All 1-pixels in the image are tested.

2.2.1. Chen-Hsu's Algorithm (CH)

This algorithm [ChH88] is a modification of Zhang-Suen's method [ZhS84] to overcome the endpoint shrinkage and to have 8-connected skeletons. Two 16×16 lookup tables are used to achieve high speed computation. Each table entry corresponds to a particular 3×3 neighborhood situation of a pixel. This neighborhood is called a *rule window*. The two tables are discussed in detail in Chapter 3. A 1-pixel is deleted if it satisfies the following deletion rules:

1. $2 \le BP \le 7$

2.a. if CSN=1 then check

a1. $n_0 \wedge n_2 \wedge n_4 = 0$

a2. $n_2 \wedge n_4 \wedge n_6 = 0$

2.b. if CSN=2 then check

b1. $(n_0 \wedge n_2 = 1)$ and $(n_4 \vee n_5 \vee n_6 = 0)$

b2. $(n_2 \wedge n_4 = 1)$ and $(n_0 \vee n_6 \vee n_7 = 0)$

3.a. if CSN=1 then check

a1. $n_0 \wedge n_2 \wedge n_6 = 0$

a2. $n_0 \wedge n_4 \wedge n_6 = 0$

3.b. if CSN=2 then check

b1. $(n_0 \wedge n_6 = 1)$ and $(n_2 \vee n_3 \vee n_4 = 0)$

b2. $(n_4 \wedge n_6 = 1)$ and $(n_0 \vee n_1 \vee n_2 = 0)$.

where the symbols $\wedge$ and $\vee$ are the logical operators AND and OR respectively.

The operation is *two-subiterative:*

(1). N-W with rules 1 and 3

(2). S-E with rules 1 and 2.

In subiteration N-W, the directions of deleting outermost pixels are north and west simultaneously, while in subiteration S-E, the directions are south and east. The 3×3 rule windows related to each subiteration are presented in Fig. 2.9 and 2.10.

```
011 001 011 000 001 011 000 000 001 011
0p0 0p1 0p1 0p1 0p1 0p1 0p0 0p1 0p1 0p1
000 000 000 001 001 001 011 011 011 011

000 000 000 001 011 010 011 000 000 000
0p0 0p0 0p1 0p1 0p1 1p0 1p0 1p0 1p0 1p0
110 111 111 111 111 000 000 010 011 100

010 011 000 000 000 001 110 111 111 111
1p0 1p0 1p0 1p0 1p1 1p1 0p0 0p0 0p1 0p1
100 100 110 111 111 111 000 000 000 001

111 111 100 110 111 100 100 100 110 111
0p1 0p1 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0
011 111 000 000 000 010 011 100 100 100

100 100 100 101
1p0 1p0 1p1 1p1
110 111 111 111
```

Figure 2.9 Rule Windows of CH (N-W subiteration)

```
011 010 001 011 000 010 001 011 000 001
0p0 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
000 000 000 000 001 001 001 001 010 010

000 000 001 000 000 001 000 000 001 000
0p0 0p1 0p1 0p0 0p1 0p1 0p0 0p1 0p1 1p0
011 011 011 110 110 110 111 111 111 100

000 000 110 111 110 111 110 111 100 110
1p0 1p0 0p0 0p0 0p1 0p1 0p1 0p1 1p0 1p0
110 111 000 000 000 000 001 001 000 000

111 111 111 100 110 111 111 111 100 110
1p0 1p1 1p1 1p0 1p0 1p0 1p1 1p1 1p0 1p0
000 000 001 100 100 100 100 101 110 110

111 100 110 111
1p0 1p0 1p0 1p0
110 111 111 111
```

Figure 2.10 Rule Windows of CH (S-E subiteration)

2.2.2. Zhang-Suen's Algorithm (ZS)

This algorithm [ZhS84] is simpler than CH and does not use a table mapping. The operation is *two-subiterative*, i.e. S-E with rules 1 and 2a and N-W with rules 1 and 3a (Section 2.2.1). The outermost layer of pixels are examined and deleted simultaneously starting from directions of south and east, then north and west. The two subiterations are implemented alternately. The deletion rules represented by rule windows for the subiteration S-E are in Fig. 2.11; the windows for the subiteration N-W are in Appendix A1 (Fig. A1.1). The number of rule windows is less than that in CH. The results are 4-connected skeletons.

```
011  001  011  000  001  011  000  000  001  000
0p0  0p1  0p1  0p1  0p1  0p1  0p0  0p1  0p1  0p0
000  000  000  001  001  001  011  011  011  110

000  000  001  000  000  000  110  111  111  111
0p0  0p1  0p1  1p0  1p0  1p0  0p0  0p0  0p1  0p1
111  111  111  100  110  111  000  000  000  001

100  110  111  111  111  100  110  111  111  100
1p0  1p0  1p0  1p1  1p1  1p0  1p0  1p0  1p1  1p0
000  000  000  000  001  100  100  100  100  110

110  111  100  110
1p0  1p0  1p0  1p0
110  110  111  111
```

Figure 2.11 Rule Windows of ZS (S-E subiteration)

2.2.3. Deutsch's Algorithm (DE)

The algorithm DE [Deu72] is similar to CH, except it uses $BP=0$ and $CSN=0$ and does not apply a table mapping. The operation is *two-subiterative*, i.e. N-E and S-W. Deleting pixels is started with the subiteration N-E, from directions of north and east, followed by the subiteration S-W, from directions of south and west. The two subiterations are implemented alternately. DE generates 8-connected skeletons. The deletion rules associated with the subiteration N-E are represented by rule windows, shown in Fig. 2.12. The other rule windows are in Appendix A1 (Fig. A1.2).

```
000 011 001 011 000 010 001 011 001 000
0p0 0p0 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p0
000 000 000 000 001 001 001 001 010 011

000 001 000 000 001 000 000 001 000 000
0p1 0p1 0p0 0p1 0p1 0p0 0p1 0p1 1p0 1p0
011 011 110 110 110 111 111 111 100 110

000 000 001 110 111 110 111 110 111 100
1p0 1p1 1p1 0p0 0p0 0p1 0p1 0p1 0p1 1p0
111 111 111 000 000 000 000 001 001 000

110 111 100 110 111 100 110 111 100 110
1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0
000 000 100 100 100 110 110 110 111 111

111 100 101
1p0 1p1 1p1
111 111 111
```

Figure 2.12 Rule Windows of DE (N-E subiteration)

2.2.4. Naccache-Shinghal's Algorithm (NS)

This algorithm [NaS84b] is quite different from the previous ones. More rule windows are used and the cases with *CSN*=3 are considered. The results are 8-connected skeletons. The operation is also *two-subiterative*, i.e. E-W and N-S. The first subiteration starts from east and west simultaneously, and then the second subiteration, from north and south. The two subiterations are performed subsequently. The rule windows related to the subiteration E-W are in Fig. 2.13, and the others are in Appendix A1 (Fig. A1.3).

```
010 001 011 000 010 001 011 000 010 001
0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
000 000 000 001 001 001 001 010 010 010

011 000 010 001 011 000 010 001 011 000
0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
010 011 011 011 011 110 110 110 110 111

010 001 011 010 011 000 010 011 000 010
0p1 0p1 0p1 1p0 1p0 1p0 1p0 1p0 1p0 1p0
111 111 111 000 000 010 010 010 011 011

011 000 010 011 000 010 011 000 010 011
1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0
011 100 100 100 110 110 110 111 111 111

110 111 110 111 110 111 110 111 110 111
0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
000 000 001 001 010 010 011 011 110 110

110 111 100 110 111 100 110 111 100 110
0p1 0p1 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0
111 111 000 000 000 010 010 010 011 011

111 100 110 111 100 110 111 100 110 111
1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0 1p0
011 100 100 100 110 110 110 111 111 111
```

Figure 2.13 Rule Windows of NS (E-W subiteration)

2.2.5. Tamura's Algorithms (T1 and T2)

These algorithms [Tam78] are based on the algorithms presented in [StR71] with several corrections [Tam78]. The results are 4-connected skeletons. There are two algorithms, *noisy-branch* (T1) and *no-branch* (T2), that use the same operation, but different rules. T2 uses 21 rule windows, while T1 uses 10 of 21 windows. The rules in T2 include the cases of $2 \le BP \le 7$, and the rules in T1 include $4 \le BP \le 7$. The operation is *four-subiterative*, i.e. S, N, W, and E. The four subiterations are implemented subsequently. The rule windows used in the subiteration S for T1 and T2 are presented in Fig. 2.14 and 2.15; respectively. The other windows are in Appendix A1 (Fig. A1.4, A1.5, A1.6, A1.7, A1.8, and A1.9).

```
011  111  111  111  111  111  110  111  111  111
0p1  0p1  0p1  1p0  1p1  1p1  1p0  1p0  1p1  1p1
001  000  001  000  000  001  100  100  100  101
```

Figure 2.14 Rule Windows of T1 (S subiteration)

```
011  001  011  000  001  011  000  110  111  111
0p0  0p1  0p1  0p1  0p1  0p1  1p0  0p0  0p0  0p1
000  000  000  001  001  001  100  000  000  000

111  100  110  111  111  111  100  110  111  111
0p1  1p0  1p0  1p0  1p1  1p1  1p0  1p0  1p0  1p1
001  000  000  000  000  001  100  100  100  100

111
1p1
101
```

Figure 2.15 Rule Windows of T2 (S subiteration)

2.3. Algorithms with Contour-Tracing

The general form of a contour-tracing algorithm is as follows.

```
do {  /* iteratively */
      for every c-pixel /* contour pixel */
            do {
                  if the neighbors of the pixel satisfy the deletion rules,
                  then change the value of the pixel from 1 to 0; /* delete */
            }
} until (no more pixel deletion);
```

2.3.1. Govindan-Shivaprasad's Algorithms (G1 and G2)

There are two algorithms [GoS87], *non-adaptive (G1)* and *shape-adaptive (G2)*. The first algorithm simply examines the c-pixels, whereas the second algorithm examines them with adjustment according to the shape of the object. The results can be 4- or 8-connected skeletons depending on the rule windows for preserving the connectivity.

Non-Adaptive Algorithm (G1)

The operation is *clockwise contour-tracing*. Every c-pixel is deleted sequentially satisfying the following deletion rules:

1. $2 \leq BP \leq 7$
2. CSN=1.

The rule windows are presented in Fig. 2.16. The results are 4-connected skeletons.

011	001	011	000	001	011	000	000	001	011
0p0	0p1	0p1	0p1	0p1	0p1	0p0	0p1	0p1	0p1
000	000	000	001	001	001	011	011	011	011
000	000	000	001	011	000	000	000	000	001
0p0	0p0	0p1	0p1	0p1	1p0	1p0	1p0	1p1	1p1
110	111	111	111	111	100	110	111	111	111
110	111	111	111	111	100	110	111	111	111
0p0	0p0	0p1	0p1	0p1	1p0	1p0	1p0	1p1	1p1
000	000	000	001	011	000	000	000	000	001
100	110	111	111	100	110	111	100	110	100
1p0	1p0	1p0	1p1	1p0	1p0	1p0	1p0	1p0	1p1
100	100	100	100	110	110	110	111	111	111

Figure 2.16 Rule Windows of G1

Shape-Adaptive Algorithm (G2)

This algorithm is similar to G1, except it uses four special windows dealing with corners 45°, 90°, and T presented in Fig. 2.17. During contour tracing, the pixels 4, 5, 9 and T are identified, and during deletion process, these pixels are examined whether or not they are deleted. Therefore, the shape of the 45-, 90- and T-corners in the object will be preserved.

9	1	1
1	0	0
1	0	0

90-corner

5	1	1	1
4	1	1	1
1	0	1	1
1	0	0	1

45-corner

1	1	1	5
1	1	1	4
1	1	0	1
1	0	0	1

45-corner

1	1	1	1
1	1	T	1
0	1	1	0
0	1	1	0

T-corner

Figure 2.17 Corner Windows

The windows in Fig. 2.17 and their rotations are tested during the thinning process by calling the following procedure ADAPTIVE [GoS87].

```
procedure ADAPTIVE {
    if 90-corner window is matched,
    then if T-corner window is matched,
        then mark the T-pixel;
        else mark the 9-pixel;
    else if 45-corner window is matched,
        then mark the 4-pixel and 5-pixel;
    if the neighbors of the 4-, 5-, 9-, or T-pixel satisfies the deletion rules,
    then change the value of the pixel from 1 to 0;
}
```

2.3.2. Pavlidis' Algorithm (PA)

The rule windows of this algorithm [Pav81] in Appendix A1 (Fig. A1.10) are similar to those of NS, but have different implementations. PA uses a *clockwise contour-tracing* operation, whereas NS uses subiterative operation. The results are 8-connected skeletons.

The operations and the rule windows of the algorithms are summarized in Table 2.1. The results of the algorithms are presented in Appendix A2.

Table 2.1 Operations and Numbers of Rule Windows

Algorithm	Operation	Number of Windows*
DE	2-subiteration	86 (43)
ZS	2-subiteration	68 (34)
CH	2-subiteration	88 (44)
NS	2-subiteration	140 (70)
T1	4-subiteration	40 (10)
T2	4-subiteration	84 (21)
G1	contour-tracing	40
G2	contour-tracing	40
PA	contour-tracing	108

* Value in brackets are the number of windows in each subiteration

2.4. Criteria

We now discuss in detail the criteria to be satisfied by a skeleton.

2.4.1. Connectivity

A most important global criterion of the thinning process is connectivity [DaP81]. From the connectedness point of view, a skeleton can be either

(1). 4-connected,

(2). 8-connected, or

(3). unconnected.

The first two can be seen clearly on stairlike skeletons shown in Fig. 2.18. It is related to thinness, which is another criterion. All possible windows to maintain the 4- and 8-connectivity are presented in Fig. 2.19 and 2.20. Skeletal pixels have an important role in generating connected skeletons, because deleting skeletal pixels will result in unconnected skeletons. In the case of two-pixel width objects, if deleting pixels is performed simultaneously, it will also result in unconnected skeletons.

Figure 2.18 Stairlike Skeletons

```
011  001  011  001  011  001  011  001  011  001
0p0  0p1  0p1  0p1  0p1  0p1  0p1  0p1  0p1  1p1
000  000  000  001  001  011  011  111  111  111

111
0p1
111
```

Figure 2.19 Rule Windows for 4-connectivity

```
010  010  010  001  011  010  010  001  001  011
0p1  0p1  0p1  0p1  0p1  0p1  0p1  0p1  1p1  0p1
000  001  010  010  010  011  110  110  010  110

010  001  011  010  011  011
0p1  1p1  1p1  1p1  1p1  1p0
111  011  100  101  101  111
```

Figure 2.20 Rule Windows for 8-connectivity

2.4.2. Thinness

The 4- and 8-connectedness are related to thinness. The skeleton should consist of 1-pixel wide lines, which is 8-connected (Fig. 2.18). This skeleton can be found by successively deleting contour pixels with the rule windows presented in Fig. 2.19 and 2.20. If the rule windows in Fig. 2.20 are not included, the skeleton will be 4-connected and looks 2-pixels wide lines in some cases (Fig. 2.18).

2.4.3. Position

An ideal skeleton should lie along the center of the object. The center position can be obtained by a symmetric deletion from all directions (N, W, S, E) starting from the outermost pixels. An asymmetric deletion results in the position of the skeleton not being in the center.

2.4.4. Stability

The stability of a skeleton means that the endpoint pixels in the skeleton are not deleted if further iterations are performed. Otherwise, the skeleton will shrink. The following examples (Fig. 2.21) show the endpoint shrinkage. Stability is related to the value of BP. The deletion rules of most algorithms include $2 \leq BP \leq 7$, i.e. a 1-pixel is deleted from the object if its BP is between 2 and 7. The 4-connected skeletons are not stable as long as $BP=2$ is included in the rule [Lu85]. In the case of 8-connected skeletons, if $BP=1$ is included in the rules, the skeletons are not stable.

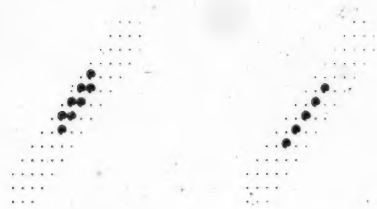

Figure 2.21 Endpoint Shrinkage Skeletons

2.4.5. Shape-Adaptive

The shape of an object is also a global property. The skeleton should preserve the shape of the original object such that misrecognition will not happen. A significant example of not preserving the original shape is shown in Fig.2.22. The original object is 'B', but the skeleton looks like '8'.

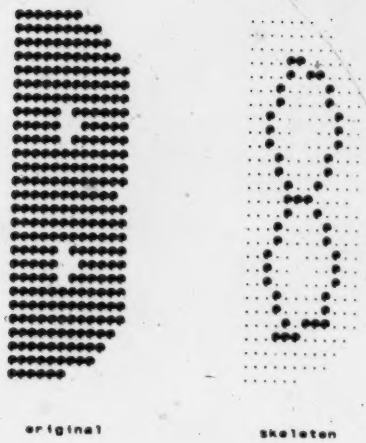

Figure 2.22 Non-shape Adaptive Skeleton

2.4.6. Noisy Branch

The thinning process should generate a skeleton without noisy branches. A skeleton with noisy branches does not preserve the shape of the original object, and some parts of the skeleton are not in the center position. This may be caused by deletion rules which delete object pixels with $4 \leq BP \leq 7$ or the rule windows of algorithm T1 shown in Fig. 2.14. This kind of skeleton is not desirable. Fig. 2.23 shows two skeletons with and without noisy branches.

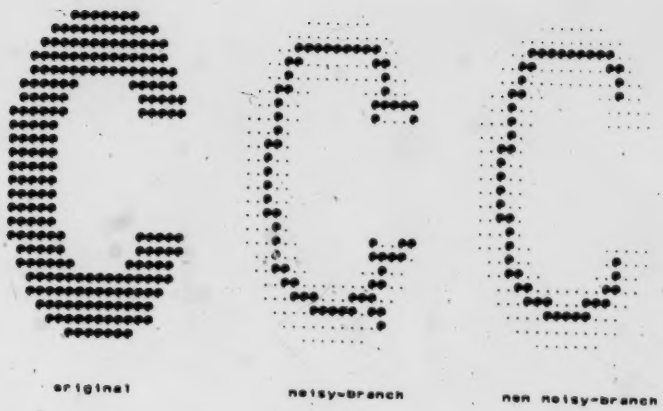

Figure 2.23 Noisy-branch and Non Noisy-branch Skeletons

2.4.7. Sensitivity

Skeletons should be immune to noise on an object which very often result in unnecessary branches and shape distortion. Skeletons should also be unique, no matter which point and direction the thinning process is started. This section discusses the sensitivity of the algorithms to noise, starting point and direction of deleting pixels.

Noise

Noise can be pixels in the contour of an object which should not belong to the object. Usually, skeletons are affected by noise. Small changes on the object contour result in different skeletons. Fig. 2.24 and 2.25 show the objects with noise and the skeletons of the objects.

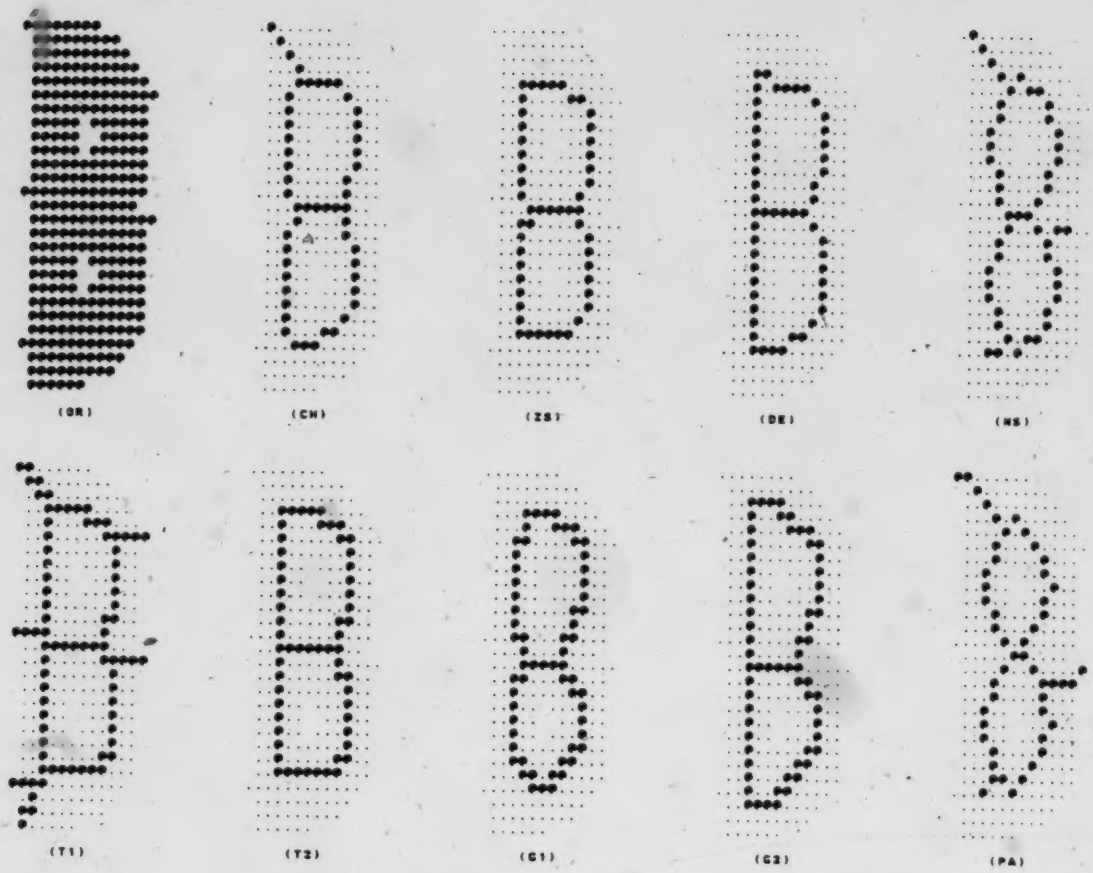

Figure 2.24 Skeletons of 'B' with Noise

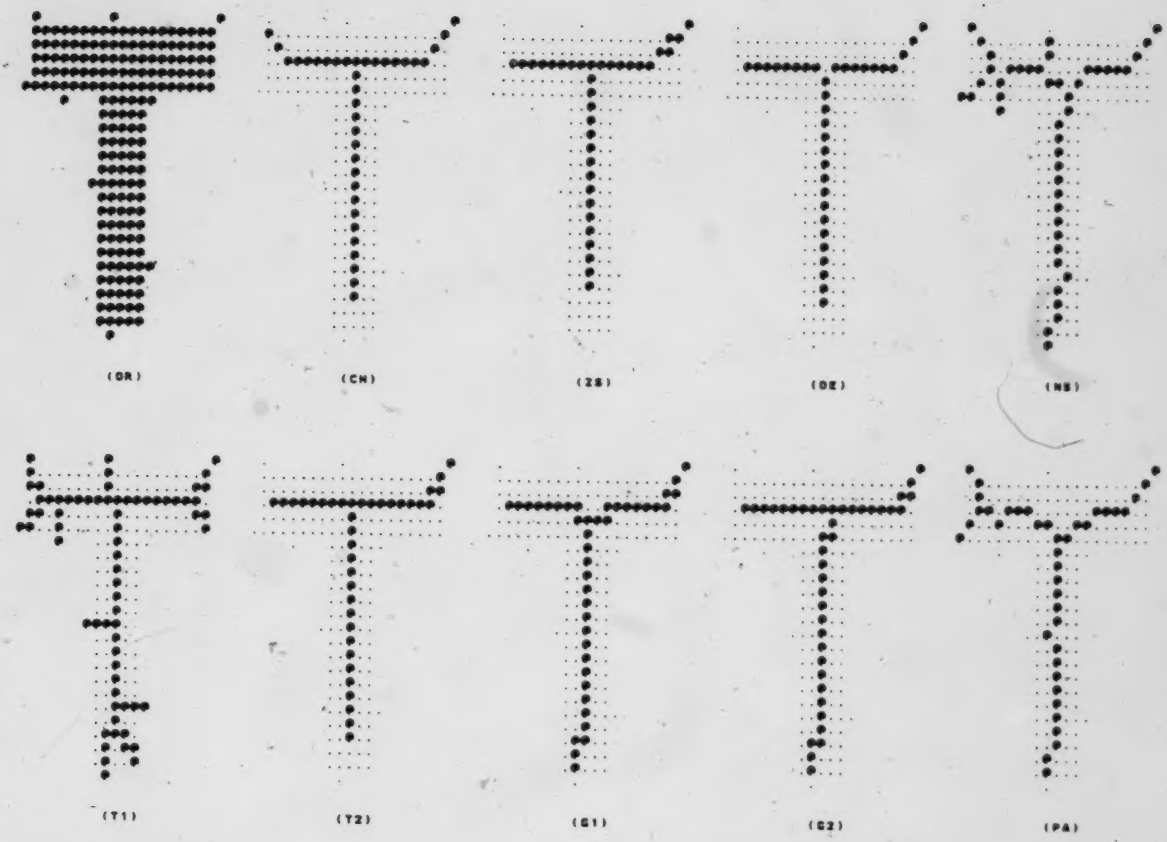

Figure 2.25 Skeletons of 'T' with Noise

In Fig. 2.24 five pixels in the contour of the original object 'B' are considered as noise. The skeletons produced by CH, NS, T1 and PA are affected by noise. Noisy branches are on the skeletons such that the skeletons are quite different from those of the object without noise (Appendix A2). In this case, these algorithms are sensitive to the noise. The skeletons produced by ZS, DE, T1, G1 and G2 are the same as those in Appendix A2.

In Fig. 2.25 nine pixels are considered as noise on the object 'T'. The noise affects the shape of the skeletons. The skeletons contain noisy branches and the shape of the object is not preserved. If there are no noisy branches, the shapes of the skeletons are similar to those of the skeletons in Appendix A2, except the skeletons generated by the algorithms NS, G1, G2 and PA.

In general, thinning algorithms are sensitive to noise. Algorithms with contour-tracing (G1, G2, PA) are more sensitive than the subiterative algorithms, except algorithms NS and T1. Algorithm PA is more sensitive than G1 and G2. Noisy branches and distortion should be minimized [Kwo88]. Smoothing is one way to overcome the problem [ChS86]. Noise elimination does not imply endpoint shrinkage.

Starting Points and Directions

There is no general rule of starting the thinning process. In subiterative algorithms, any permutation of directions (N, W, S, E) can be implemented for deleting pixels. In algorithms with contour-tracing, deleting pixels can be started from any contour pixel in a clockwise (CW) or counter clockwise (CCW) direction. Several results are quite different. It is indicated that the skeletons of an object are not unique [DaP81], depending on the way one performs the thinning process. Besides the skeletons in Appendix A2, several skeletons in Appendix A3 are generated by algorithms CH, ZS, DE, T2, NS, G2, and PA implemented in various ways.

The algorithms CH, ZS and DE are implemented in four different ways. The skeletons of 'T' are similar, but the skeletons of 'B' are different. CH is implemented in the ways of NW-SE

(the results are in Appendix A2), SE-NW, SW-NE, and NE-SW. The skeleton of 'B' in Fig. A3.1 results from the second implementation (SE-NW) and is better than the other skeletons. The skeleton maintains the T-corner of 'B', but does not preserve 90-corners. ZS is implemented in similar ways, i.e. SE-NW (the results are in Appendix A2), NW-SE, NE-SW, and SW-NE. The skeletons of 'B' in Fig. A3.2 have different distortions. The algorithm is very sensitive to starting points and directions of deleting pixels. Similarly, DE is implemented in four ways, i.e. NE-SW (the results are in Appendix A2), SW-NE, SE-NW, and NW-SE. In Fig. A3.3 the skeleton of 'B' of the third implementation (SE-NW) is better than the others. The skeleton preserves the T- and 90-corners of 'B'.

The algorithm T2 is performed in six different ways, i.e. SNWE (the results are in Appendix A2), SENW, NWSE, SWNE, NESW, and SNEW. Four different skeletons of 'B' are generated. The T- and 90-corners are preserved, but the skeletons have different distortions. The skeletons of 'T' are similar, and the shape of the original object 'T' is maintained. The algorithm T1 will give similar results, if the six different ways are implemented. The algorithm NS is performed in two different ways, i.e. EW-NS (the results are in Appendix A2) and NS-EW. Both results are similar, but some parts of the skeletons are different.

The algorithms G2 and PA are implemented in the following five ways:

1. clockwise with starting point-1 (CW-1)
2. clockwise with starting point-2 (CW-2)
3. clockwise with starting point-20 (CW-20)
4. counter clockwise with starting point-2 (CCW-2)
5. counter clockwise with starting point-20 (CCW-20).

The results of the first implementation are in Appendix A2, and the others are in Appendix A3. The skeletons of 'T' generated by G2 are the same, and those generated by PA are also the same, except the skeletons resulting from CW-20 contain noisy branch (Fig. A3.6 and A3.7). The skeleton of 'B' generated by G2 with CW-1, CW-2, and CCW-2 are the same, but are different

from the skeletons implemented with CW-20 and CCW-20. Similarly, the skeletons of 'B' generated by PA with CW-1, CW-2, and CCW-2 are the same, but are different from the skeletons produced with CW-20 and CCW-20. The algorithm G1 will produce similar results to the results of G2, if the five ways are performed.

2.5. Comparisons

Experiments are conducted on 26 English letters. The nine algorithms discussed above are applied to these characters. The variation in letter forms is not examined. In this section, we compare these algorithms based on the skeletons generated, the criteria satisfied, and time complexity.

2.5.1. Skeletons and Criteria Satisfied

Connectivity

All skeletons in Appendix A2 are connected as well as the original objects. The skeletons generated by CH, DE, NS, and PA are 8-connected, while the other skeletons are 4-connected. The rules ($CSN \geq 1$ or rule windows in Fig. 2.19 and 2.20) of the algorithms CH, DE, NS, and PA are used to produce 8-connected skeletons. The rules ($CSN = 1$ or rule windows in Fig. 2.19) of the algorithms ZS, T1, T2, G1, and G2 result in 4-connected skeletons. In general, the algorithms maintain the connectivity of original objects, except ZS produces skeletons which are not really 4-connected.

Center Position

Most of the skeletons are in the center of the original objects, except the skeletons generated by T1 that consist of noisy branches. The center position results from the iterative process which is implemented starting from the outermost layer of pixels. In subiterative algorithms (CH, ZS,

DE, NS, T1, T2), the outermost layer of pixels are deleted subsequently corresponding to the number of subiterations per iteration. In algorithms with contour-tracing (G1, G2, PA), outer and inner contour pixels are deleted alternately, such as in characters 'B', 'D', 'O', 'P', 'Q' and 'R'. The operations and the ways of performing algorithms have an important role in satisfying this criterion. The algorithms maintain the center position of skeletons.

Noisy Branch

The noisy-branch skeletons are only generated by T1. Most parts of the skeletons are in the center position. If the noisy branches are removed, the skeletons will be the same as those produced by T2. Both algorithms use the same operation but different rules. T1 includes the value of $4 \leq BP \leq 7$ in the rules, whereas T2 applies the value of $2 \leq BP \leq 7$.

Thinness

Thinness of 1-pixel width is satisfied by most skeletons, except the 4-connected skeletons that look like 2-pixels in width at some places, such as skeletons produced by the algorithms ZS, T1, T2, G1 and G2. The other algorithms produce 1-pixel width and 8-connected skeletons. It is indicated that thinness and connectedness are closely related. It can be seen that the 1-pixel width is controlled by the rules for generating 8-connected skeletons.

Shape Adaptive

Most skeletons produced by DE, T2 and G2 maintain the shape of the original objects, even though several skeletons are not shape adaptive, such as the skeletons of 'A', 'O' and 'Q' produced by G2, and the skeleton of 'S', which looks like '5', produced by T2. Most skeletons generated by the other algorithms do not fully preserve the object shape, especially skeletons generated by T1, NS, G1, and PA. Some parts of the skeletons are distorted. In the worst case, the skeletons of 'B' generated by NS, G1, and PA look like an '8', and all skeletons generated by T1

contain noisy branches.

Stability

Most skeletons are stable, except some 4-connected skeletons produced by ZS, G1, G2, and T2. In some cases, the rules for generating 4-connected skeletons cause endpoint shrinkage, such as skeletons of characters 'C', 'J', 'K', 'S', 'X', and 'Y'. Generally, the algorithms that produce 4-connected skeletons generate unstable skeletons, and the algorithms that produce 8-connected skeletons generate stable skeletons.

2.5.2. Time Complexity

Time complexity is based on the size of an image and the number of 1-pixels in the image [Kwo88]. For subiterative algorithms, in a subiteration, all pixels in the image are scanned and each 1-pixel is examined by deletion rules. For algorithms with contour-tracing, in an iteration, the image is also scanned to identify 1-pixels and only c-pixels are examined. Therefore, the number of time units is composed of the time for scanning the image and for evaluating the deletion rules.

If the size of the image is $n \times n$, scanning the image requires n^2 time units. In an iteration, 2-subiterative algorithms need $2n^2$ time units, 4-subiterative algorithms need $4n^2$, and algorithms with contour-tracing need n^2. The time complexity of all algorithms is $O(n^2)$. However, algorithms with contour-tracing can achieve a time reduction because they only examine c-pixels of an object [Kwo88].

Moreover, every algorithm requires time to evaluate the deletion rules. If it is assumed that the time units used for additions, comparisons, and logical operators are the same, Table 2.2 presents the number of time units for the algorithm CH without table mapping. Evaluating the rules for every 1-pixel requires at most 15 additions, 1 division, 10 comparisons, and 10 logical

operations. The time requirement for CH with table mapping is discussed in Chapter 3. Table 2.3 presents the time requirement for all algorithms, including their average execution time which is calculated from the time used to generate the skeletons of some characters on a Vax-11/780.

Table 2.2 Computation Time for Evaluating CH Rules

Step Number	Addition	Division	Comparison	Logical Operation
1	7	0	2	0
2a or 3a	8	1	3	4
2b or 3b	0	0	5	6

Table 2.3 Time Requirement of Algorithms

Algorithm	Computation time to evaluate rules per pixel				Average Execution Time (seconds)
	Addition	Division	Comparison	Logical Operation	
CH	15	1	10	10	1.18
ZS	15	1	5	4	1.24
DE	15	1	11	14	1.20
NS	15	1	20	0	1.31
T1	15	1	3	0	1.81
T2	15	1	3	0	1.77
G1	15	1	3	0	1.37
G2	15	1	3	0	1.38
PA	15	1	20	0	1.42

Algorithms with 4-subiterative operations, such as T1 and T2, require more time. The algorithms CH, ZS and DE require about the same amount of time, but ZS needs more iterations to complete the thinning process. Similarly, G1 also needs more iterations than G2 and PA. About 25% of the time used by the algorithms G1, G2 and PA is used for the contour-tracing process.

Chapter 3

A Table Reformulation

Several existing methods can be reformulated in a table lookup framework to achieve flexibility as the table can be an experimental tool. This chapter discusses a table reformulation and presents several approaches to implement the reformulated table. This new table is a modification of the lookup tables originally presented in [ChH88]. The operation used by the new approaches can be either 4-subiterative or contour tracing.

In Section 3.1, the lookup tables used in CH are presented. The reformulated table for the existing methods is discussed in Section 3.2, and the results of the new methods are compared in Section 3.3.

3.1. Lookup Tables

Table mapping used in CH [ChH88] is a way of implementing the thinning process to achieve high speed computation. In this approach, two 16×16 tables are used to store the deletion rules for two subiterations (S-E and N-W). Each table entry contains a decision value, 'N' for Not-deleted or 'D' for Deleted, to decide whether or not a pixel is deleted. These lookup tables are shown in Table 3.1 and 3.2. Each table requires 256 bytes. The total storage is 512 bytes.

A table entry is accessed directly using the 1-byte table index which is calculated from the values of the 8-neighbors ($n_7n_6n_5n_4n_3n_2n_1n_0$) of a pixel under examination. The sequence of the neighbors is considered as an 8-bit address where the value 'N' or 'D' is stored. The first 4 bits ($n_7n_6n_5n_4$) is the first index in the table, and the other 4 bits ($n_3n_2n_1n_0$) is the second index. The following example illustrates the 8-neighbors of pixel p (Fig. 3.1) mapped into the address 00110110. The first index is 0011 (3 in hexadecimal) and the second index is 0110 (E in

hexadecimal). According to the table entry in Table 3.1, the pixel is deleted.

0	0	1
0	p	1
1	1	0

Figure 3.1 Values of 8-neighbors

Table 3.1 Deletion Rules for S-E Subiteration

First Index	Second Index															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	N	N	N	D	N	D	D	D	N	N	N	N	D	D	D	D
1	N	N	N	N	D	N	D	N	D	N	N	N	D	N	D	N
2	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
3	D	N	N	N	D	N	D	N	D	N	N	N	D	N	D	N
4	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
5	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
6	D	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
7	D	N	N	N	N	N	N	N	D	N	N	N	N	N	N	N
8	N	D	N	D	N	D	N	D	N	N	N	N	N	D	N	D
9	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
A	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
B	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
C	D	D	N	D	N	N	N	D	N	N	N	N	N	N	N	D
D	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
E	D	D	N	D	N	N	N	D	N	N	N	N	N	N	N	D
F	D	D	N	D	N	N	N	N	D	D	N	D	N	N	N	N

Table 3.2 Deletion Rules for N-W Subiteration

First Index	Second Index															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	N	N	N	D	N	N	D	D	N	N	N	N	D	N	D	D
1	N	N	N	N	N	N	N	N	D	N	N	N	D	N	D	D
2	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
3	D	N	N	N	N	N	N	N	D	N	N	N	D	N	D	D
4	N	D	N	D	N	N	N	N	N	N	N	N	N	N	N	N
5	D	N	N	N	N	N	N	N	D	N	N	N	N	N	N	N
6	D	D	N	D	N	N	N	N	N	N	N	N	N	N	N	N
7	D	N	N	N	N	N	N	N	D	N	N	N	D	N	D	N
8	N	D	N	D	N	N	N	D	N	N	N	N	N	N	N	D
9	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D
A	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
B	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	D
C	D	D	N	D	N	N	N	N	N	N	N	N	N	N	N	N
D	D	N	N	N	N	N	N	N	D	N	N	N	N	N	N	N
E	D	D	N	D	N	N	N	N	N	N	N	N	N	N	N	N
F	D	N	N	N	N	N	N	N	D	N	N	N	D	N	D	N

The algorithm CH using table mapping is as follows.

```
do { /* iteratively */
     for every subiteration,  /* 2 subiterations */
          do {
               for every 1-pixel,
                    find the sequence of 8-neighbors;
                    look up the corresponding table;
                    if the value in the address is 'D',
                         then mark the 1-pixel by 'd';
          }
          do {
               for every pixel marked 'd',
                    change the value of the pixel from 1 to 0; /* delete */
          }
} until (no more pixel deletion);
```

3.2. Reformulation

The lookup table is reformulated to consist of only the 'D' entries for certain algorithms, and so the table is flexible. It is easily modified to perform the algorithms, and because of the small number of 'D' entries, the table makes experiments quite easy.

The algorithm CH needs 512 bytes to store two 16×16 tables. Each table contains 44 entries 'D' and 212 entries 'N'. In fact, 66 of 88 'D' entries of the two tables are different, and the values 'N' are not necessary. These 60 addresses correspond to 60 different neighbor windows (Fig. 3.2). Considering the similarity and symmetry of a window with its 90°, 180°, and 270° rotations, the 4 windows can be combined to 1 window, and so the 60 windows can be combined to 15 windows (Fig. 3.3) to achieve a 75% reduction. Therefore, only 15 values 'D' are used in a one-dimension table, and the storage is reduced to 15 bytes.

```
011  000  000  100  001  000  000  110  011  000
0p0  0p1  0p0  1p0  0p1  0p0  1p0  0p0  0p1  0p1
000  001  110  000  000  011  100  000  000  011

000  110  001  000  100  111  011  000  100  111
1p0  1p0  0p1  0p0  1p0  0p0  0p1  0p1  1p0  1p0
110  000  001  111  100  000  001  111  110  000

001  000  110  111  011  000  110  111  001  100
0p1  1p0  1p0  0p1  0p1  1p1  1p0  1p1  0p1  1p0
011  111  100  000  011  111  110  000  111  111

111  111  011  100  111  111  001  110  111  111
1p0  0p1  0p1  1p1  1p0  1p1  1p1  1p0  1p1  0p1
100  001  111  111  110  001  111  111  100  011

101  111  111  111  010  000  000  010  010  000
1p1  1p0  1p1  0p1  0p1  0p1  1p0  1p0  0p1  0p1
111  111  101  111  000  010  010  000  001  110

100  011  001  000  010  110  001  100  011  110
1p0  1p0  0p1  1p0  1p0  0p1  0p1  1p0  1p0  0p1
010  000  010  011  100  000  110  011  100  001
```

Figure 3.2 60 Different Neighbor Windows in CH

```
011  001  011  001  011  001  011  001  011  001
0p0  0p1  0p1  0p1  0p1  0p1  0p1  0p1  0p1  1p1
000  000  000  001  001  011  011  111  111  111

101  010  010  001  001
1p1  0p1  0p1  0p1  0p1
111  000  001  010  110
```

Figure 3.3 15 Different Neighbor Windows in CH

However, in order to examine the neighborhood of a pixel and decide if there is a table entry corresponding to it, the following procedure SEQUENCE is proposed. This procedure checks the neighboring pixels and returns a sequence pattern which is used to access the table. Consider the 8-neighbors in Fig. 2.2. The pixels n_0, n_2, n_4 and n_6 are traced until the first value 0 is found. If the value 0 is in one of the 4 pixels, then starting from that pixel, the pixels in the neighbor path are traced clockwise until the first value 1 is found. If the value 1 is in n_1, n_3, n_5, or n_7, the sequence pattern is 'abcdef10'; otherwise, the sequence pattern is 'abcdefg1'. The values of 'a', 'b', 'c', 'd', 'e', 'f' and 'g' are the values (0 or 1) of the next pixels in the neighbor path after the first value 1. If the value 0 is not found in the 4 pixels, the pixel p in the neighbor window is not deleted.

Procedure SEQUENCE:

```
trace pixels n0, n2, n4 and n6 until the first value 0 is found;
if the value 0 is not found,
        exit;
else {
        trace the next pixels in the neighbor path until the first value 1 is found;
        if no pixel has value 1,
                the sequence is 00000000;
        else {
                if the value 1 is in n1, n3, n5, or n7,
                        the sequence is 'abcdef10';
                else
                        the sequence is 'abcdefg1';
                        /* a, b, c, d, e, f, and g are the values (0 or 1) of pixels after the first value 1
                        in the neighbor path */
        }
}
exit;
```

Table 3.3 shows the computation time required for finding a table index. The time is measured based on the number of steps used to load neighbor pixels and to find the corresponding sequence pattern. The procedure SEQUENCE requires 16 time units, while the procedure used in CH requires 14 units.

Table 3.3 Computation Time for Finding an Index

Computation	Procedure CH	Procedure SEQUENCE
Load pixels	8	8
Compare	0	5
Shift	6	3
Total	14	16

The operation used in the algorithm CH is not appropriate if the table containing 15 entries is used. The results do not satisfy the criteria. The corresponding operation is 4-subiterative or contour tracing. Two modified algorithms are as follows.

Algorithm with Four-subiterative

```
do { /* iteratively */
    for every subiteration /* 4 subiterations */
        do {
            for every 1-pixel,
                do {
                    call procedure SEQUENCE;
                    if a sequence pattern is found,
                        look up table;
                        if the sequence pattern is matched,
                            mark the 1-pixel by 'd';
                }
        }
        do {
            for every pixel marked 'd',
                change the value of the pixel from 1 to 0; /* delete */
        }
} until (no more pixel deletion);
```

In 4-subiterative operations, contour pixels are deleted simultaneously in each subiteration, and the 4 subiterations are implemented sequentially until no more pixel is deleted. By calling the procedure SEQUENCE and looking up the table, every 1-pixel is tested whether or not it is deleted.

Algorithm with Contour-tracing

```
do { /* iteratively */
    for every c-pixel,
        do {
            call procedure SEQUENCE;
            if a sequence pattern is found,
                look up table;
                if the sequence pattern is matched,
                    change the value of the pixels from 1 to 0; /* delete */
        }
} until (no more pixel deletion);
```

In contour-tracing operation, contour pixels are deleted sequentially until no more pixel is deleted. By calling the procedure SEQUENCE and looking up the table, every c-pixel is tested whether or not it is deleted.

The rules of the other algorithms, ZS, DE, NS, T1, T2, G1, G2 and PA, (Chap.2) can also be represented by sequence patterns. The sequence patterns and deletion values 'D' for every algorithm are presented in Table 3.4. The reduction of the number of entries is summarized in Table 3.5.

Table 3.4 Sequence Patterns and Decision Values 'D'

No.	Sequence Pattern	Algorithm								
		CH	ZS	DE	NS	T1	T2	G1	G2	PA
0	00000000			D						
1	00000011	D	D	D	D		D	D	D	D
2	00000110	D	D	D	D		D	D	D	D
3	00000111	D	D	D	D		D	D	D	D
4	00001110	D	D	D	D		D	D	D	D
5	00001111	D	D	D	D	D	D	D	D	D
6	00011110	D	D	D	D	D	D	D	D	D
7	00011111	D	D	D	D	D	D	D	D	D
8	00111110	D	D	D	D	D	D	D	D	D
9	00111111	D	D	D	D	D	D	D	D	D
10	01111110	D	D	D	D	D	D	D	D	D
11	00000101	D			D					D
12	00010110	D		D	D					D
13	00001101	D		D	D					D
14	00010101				D					D
15	00010111				D					D
16	00011101				D					D
17	00110101				D					D
18	00110110	D		D	D					D
19	00110111				D					D
20	00111101				D					D
21	01010110				D					D
22	11010110				D					D
23	01011110				D					D
24	01110110				D					D
25	11110110				D					D
26	11011110				D					D
27	11111110	D		D	D	D	D			D

Table 3.5 Numbers of Table Entries 'D'

Algorithm	Number of Entries 'D'	
	Original	Reformed
CH	60	15
ZS	40	10
DE	57	15
NS	108	27
T1	28	7
T2	44	11
G1	40	10
G2	40	10
PA	108	27

Table 3.4 shows one table containing 28 entries 'D' for nine algorithms. Each algorithm can use the table by simply setting the corresponding patterns. Moreover, the table can be easily modified to implement other algorithms. Because of its flexibility, the table can be used as an experimental tool to find a set of sequence patterns as deletion rules for an algorithm. It seems that this table brings many algorithms together and makes the comparison and analysis quite easy.

The nine algorithms include the sequence patterns 1 to 10, except algorithm T1. This algorithm includes sequence patterns 5 to 10 and 27 which correspond to the values of $4 \le BP \le 7$ and CSN=1. These sequence patterns result in a noisy-branch skeleton. The sequence patterns 1 to 10 and 27 are the rules to have 4-connected skeletons, such as the skeletons produced by the algorithms ZS, T1, T2, G1, and G2. The other sequence patterns are the addition rules mainly used to get 8-connected skeletons, such as in the algorithms DE, CH, NS, and PA.

The algorithms NS and PA include sequence patterns which result in similar skeletons. The algorithms G1 and G2 also include the same sequence patterns, but the skeletons are different, because G2 uses a special procedure to get rid of corners (Chap.2, Fig. 2.17). The algorithm ZS includes the same sequence patterns as the algorithms G1 and G2, but ZS uses 2-subiterative instead of contour-tracing operation. Sequence patterns in the algorithms DE and CH are the

same, except for one different pattern. DE uses 00000000, while CH uses 00000101. The operations are the same, and the skeletons produced are similar. The algorithm T2 includes one different pattern, i.e. 11111110, from ZS, G1, and G2. T2 uses 4-subiterative operation which is different from the operations used in the three algorithms.

The number of entries 'D' for each algorithm (Table 3.5) is reduced to about 25%. The largest number, 27, is for NS and PA. These entries cover all possible deletion rules to get 8-connected skeletons. The smallest number of entries, 7, is for T1. These entries are used to preserve 4-connectivity, but generate noisy-branch skeletons. The number of entries, 10 or 11, such as for ZS, T2, G1 and G2, preserves 4-connected skeletons without noisy branches.

3.3. Comparison

The original and reformed algorithms CH, T2 and G2 use the same deletion rules, but different implementations, i.e. apply the original and reformed lookup tables, respectively. We compare these algorithms based on the storage and time requirement, and skeletons generated.

3.3.1. Storage Requirement for Table Entries

The storage requirement for these algorithms and storage reduction are shown in Table 3.6. The largest storage reduction is from 1024 to 11 bytes used by the original and reformed T2, respectively, because the original T2 uses 4-subiterative operation which needs 4 16×16 tables. The smallest reduction is in the algorithms G2, from 256 to 10 bytes as the original G2 uses only 1 16×16 table. The original CH requires 512 bytes, while the reformed CH requires 15 bytes. In general, the reformed algorithms require less storage than the original algorithms.

Table 3.6 Storage Requirement for Table Entries

Algorithm		Storage (bytes)
CH	Original	512
	Reformed	15
T2	Original	1024
	Reformed	11
G2	Original	256
	Reformed	10

3.3.2. Time Requirement

The time requirement for the original and reformed algorithms CH, T2 and G2 are shown in Table 3.7. Time units required to find a table index and a number of indices per iteration are based on the number of computations in Table 3.3. The average execution time reported is calculated from the time used to generate the skeletons of some characters on a Vax-11/780.

Table 3.7 Time Requirement of Algorithms CH, T2 and G2

Algorithm		Finding Table Indices		Execution Time (seconds)
		per pixel	each iteration	
CH	Original	14	28b	1.05
	Reformed	16	64b	1.51
T2	Original	14	56b	1.17
	Reformed	16	64b	1.35
G2	Original	14	14c	1.14
	Reformed	16	16c	1.39

b =the number of 1-pixels in an image;
c =the number of c-pixels in an object

In general, the reformed algorithms need more processing time. Since $c < b$, the reformed G2 with contour-tracing is better than the reformed CH and T2 with 4-subiterative operation. These algorithms which use binary search need more time to find a table entry than the original ones which require only 1 time unit. The algorithms with original table mapping are faster than those with the reformed table mapping.

3.3.3. Skeletons

The skeletons generated by the reformed CH, T2, and G2 are presented in Appendix A4 (Fig. A4.1, A4.2 and A4.3). Most of them are the same as the skeletons generated by the original algorithms. Three skeletons, 'B', 'K' and 'R', produced by the reformed CH are not quite the same as those produced by the original CH, because of different operations used in both algorithms. The reformed CH which uses 4-subiterative operation preserves the T-corners of the object, whereas the original CH which uses 2-subiterative operation does not maintain the corners. The skeletons generated by the original and reformed T2 are exactly the same. Similarly, the original and reformed G2 also generate the same results.

3.4. A New Set of Sequence Patterns

The original and reformed algorithms G2 generate the same 4-connected skeletons. A set of sequence patterns in Table 3.4 can be applied to the reformed G2 in order to get 8-connected skeletons. The patterns are 10 patterns used in G2, 00000101, 00010110, 00001101, 00010101, and 11111110.

A special treatment is used to incorporate the patterns 00000101 to preserve 90-corners and 00010101 to preserve T-corners, using the windows '101' and '10101' shown in Fig. 3.4. If the neighbors of p in either window '101' or '10101' are matched, the pixel p in the window '101' is

deleted if the value of a or b is not 0, and the pixel p in the window '10101' is deleted if at least one of the values a, b, and c is 0; otherwise, the pixel p is not deleted. In order to preserve 45-corners, the procedure used in the original algorithm G2 (Chap.2) is applied.

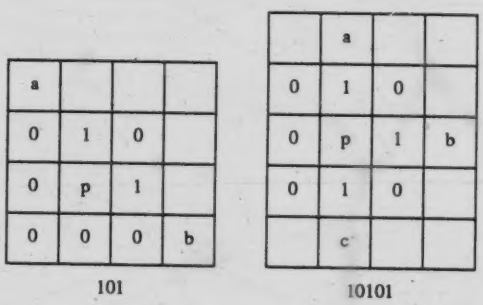

Figure 3.4 Possible Windows for 90- and T-Corners

The skeletons generated by the reformed G2 with the new set of sequence patterns are presented in Fig. 3.5. The skeletons are similar to the skeletons produced by the original G2 (Appendix A2), but have different connectivity. The skeletons generated by the original G2 are 4-connected, whereas those in Fig. 3.5 are 8-connected.

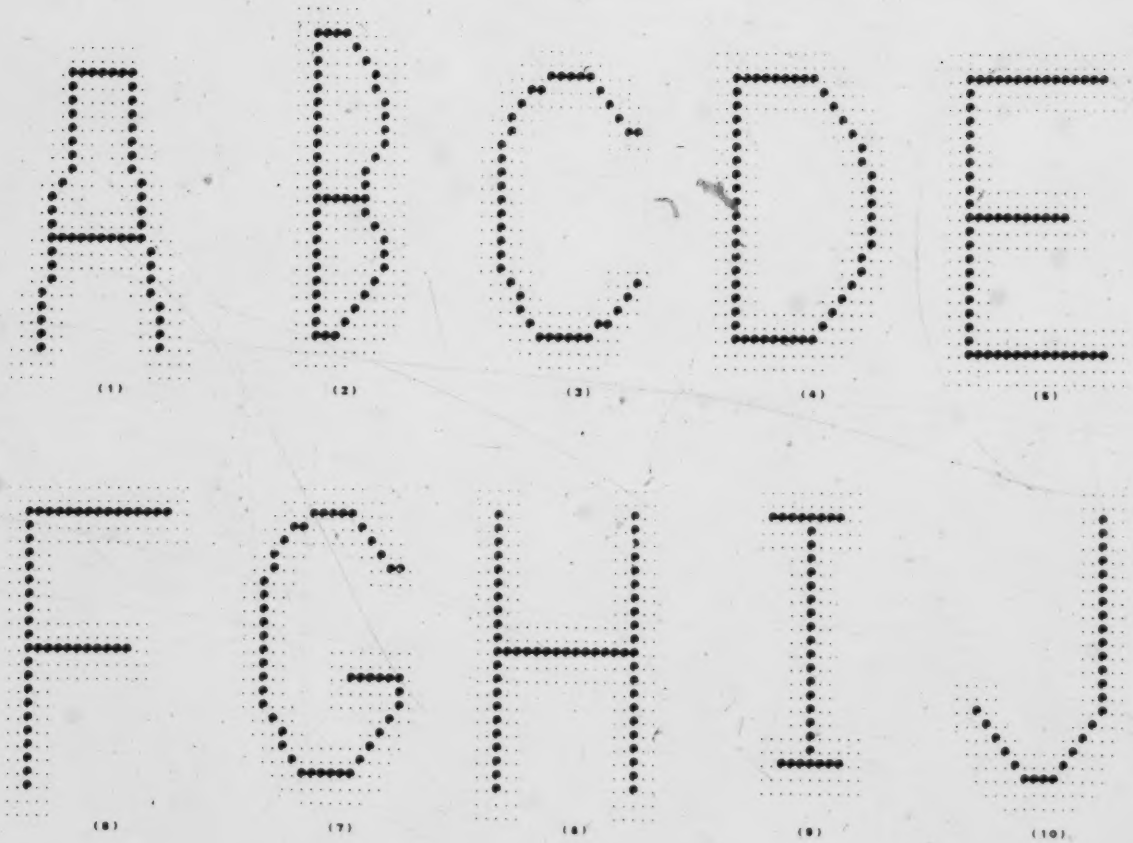

Figure 3.5 Skeletons produced by New Set

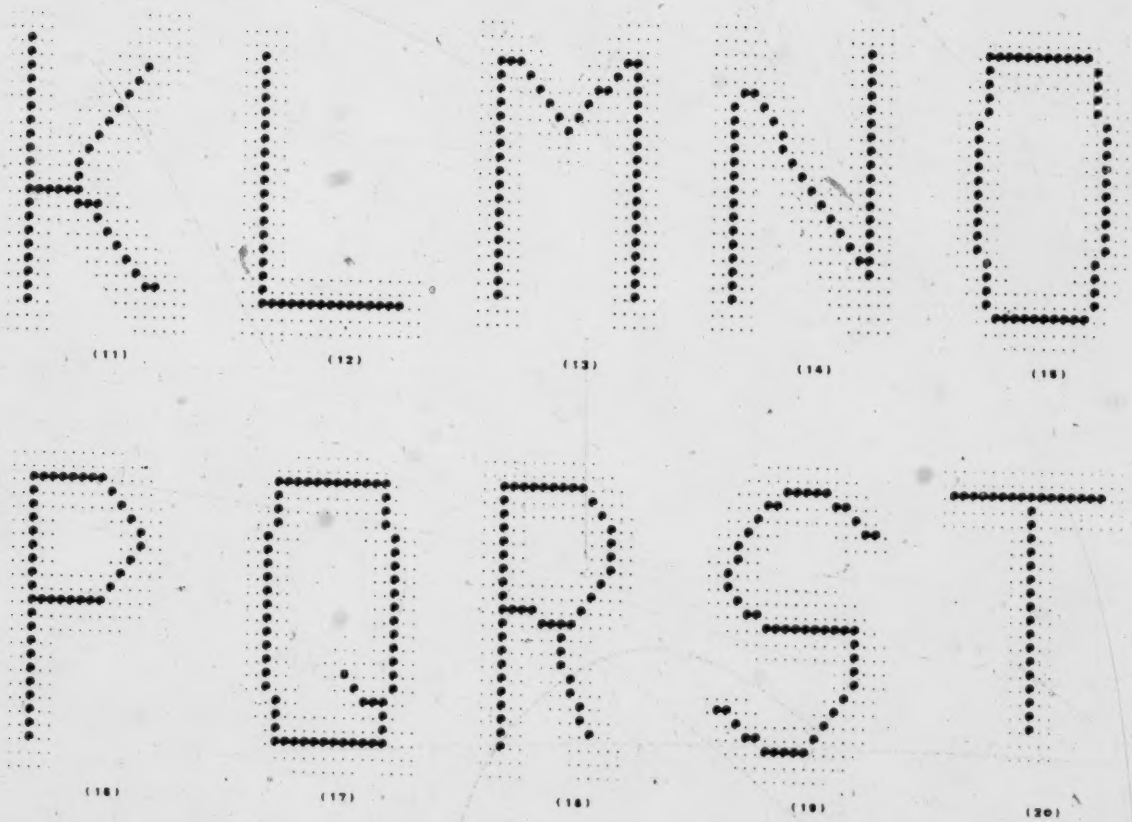

Figure 3.5. (cont.)

Figure 3.5. (cont.)

3.5. Summary

The reformed table lookup with the procedure SEQUENCE is used to achieve flexibility, and the table can be used as an experimental tool. The flexibility of the reformed lookup table allows to modify deletion rules easily and to find a set of rules for an algorithm. In terms of storage requirement for table entries, the reformed algorithms are better. The operation used by these algorithms is 4-subiterative or contour-tracing. Finding table indices per iteration by contour-tracing operation is faster than that by 4-subiterative operation. However, the original algorithms are faster. The skeletons generated by the reformed algorithms are the same as those generated by the original ones if both use the same operation.

Chapter 4

A Proposed Thinning Algorithm

Some results of the existing algorithms do not fully satisfy the criteria described in Section 2.4, especially shape adaptive criterion. The skeletons are distorted in several cases. This chapter introduces a new thinning algorithm to have skeletons which satisfy the criteria. This algorithm uses a contour-tracing operation, and applies some windows and window patterns presented in Section 4.1. Section 4.2 discusses the algorithm in detail including its sensitivity and time complexity. In Section 4.3, the algorithm is compared to the existing algorithms based on skeletons generated, criteria satisfied, and time and storage requirements.

4.1. Windows and Patterns

Some 2×2 and 3×3 *windows* and *window patterns* including special windows and patterns concerning corners about 45° are defined. These windows and patterns correspond to a pair of codes, *pcode*, that consists of *incode* and *outcode* of every c-pixel (Chap.2). The algorithm involves a mapping of pcodes to the window patterns.

4.1.1. Pcodes

There are 40 possible pcodes used in the algorithm. Ten of them are 00, 01, 02, 06, 07, 10, 11, 12, 13, and 17, shown in Fig. 4.1. The others are the 90°, 180°, and 270° rotations of the ten pcodes. Pcodes together with contour pixels are determined during the contour-tracing process. A pcode of a c-pixel determines an appropriate window and its pattern for the c-pixel.

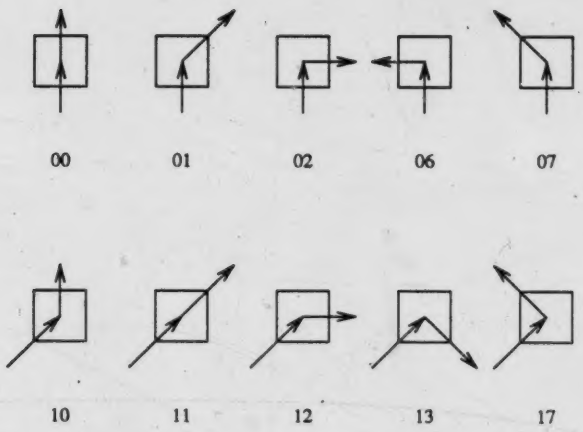

Figure 4.1 Pcodes

4.1.2. Windows

Depending on the position of a c-pixel under examination in a window, eight possible 3×3 windows and four 2×2 windows are used. Fig. 4.2 shows three possible windows, i.e. two 3×3 windows and one 2×2 window. The other windows are the 90°, 180°, and 270° rotations of the three windows. All pixels in a window are black. A window of a c-pixel is applied according to the pcode of the c-pixel.

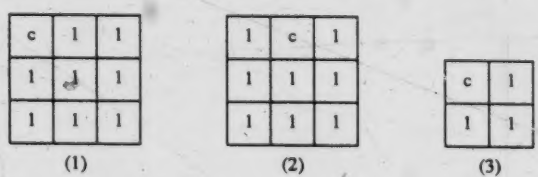

Figure 4.2 Windows of c-pixel

4.1.3. Window Patterns

Pcodes 00, 01, 02, 06, 07, 10, 11, and 12 have their window patterns presented in Fig. 4.3 (the pixel labels 'c' are for c-pixels, 'b' for next contour pixels, and 'd' for deletable pixels). The pcode 13 has window patterns to get rid of corners about 45°. Fig. 4.4 shows four possible 5×5 windows for this pcode with the corresponding two 3×3 and two 2×2 window patterns. The pcode 17 has similar window patterns which are the 90° rotations of the patterns for the pcode 13. The window patterns of the other pcodes are the 90°, 180°, and 270° rotations of those of the ten pcodes.

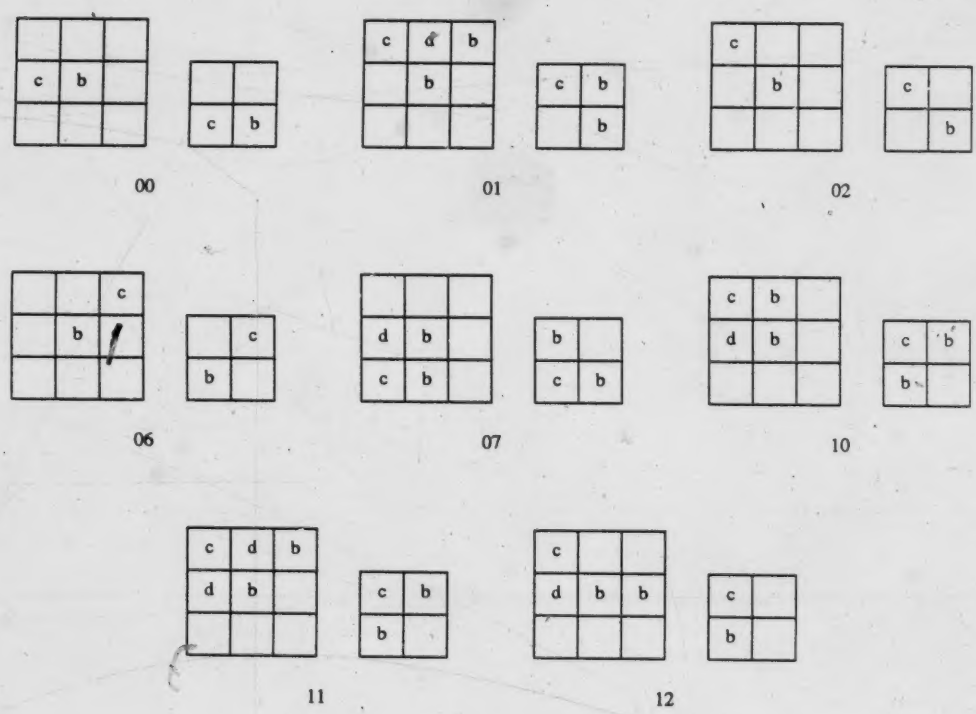

Figure 4.3 Window Patterns

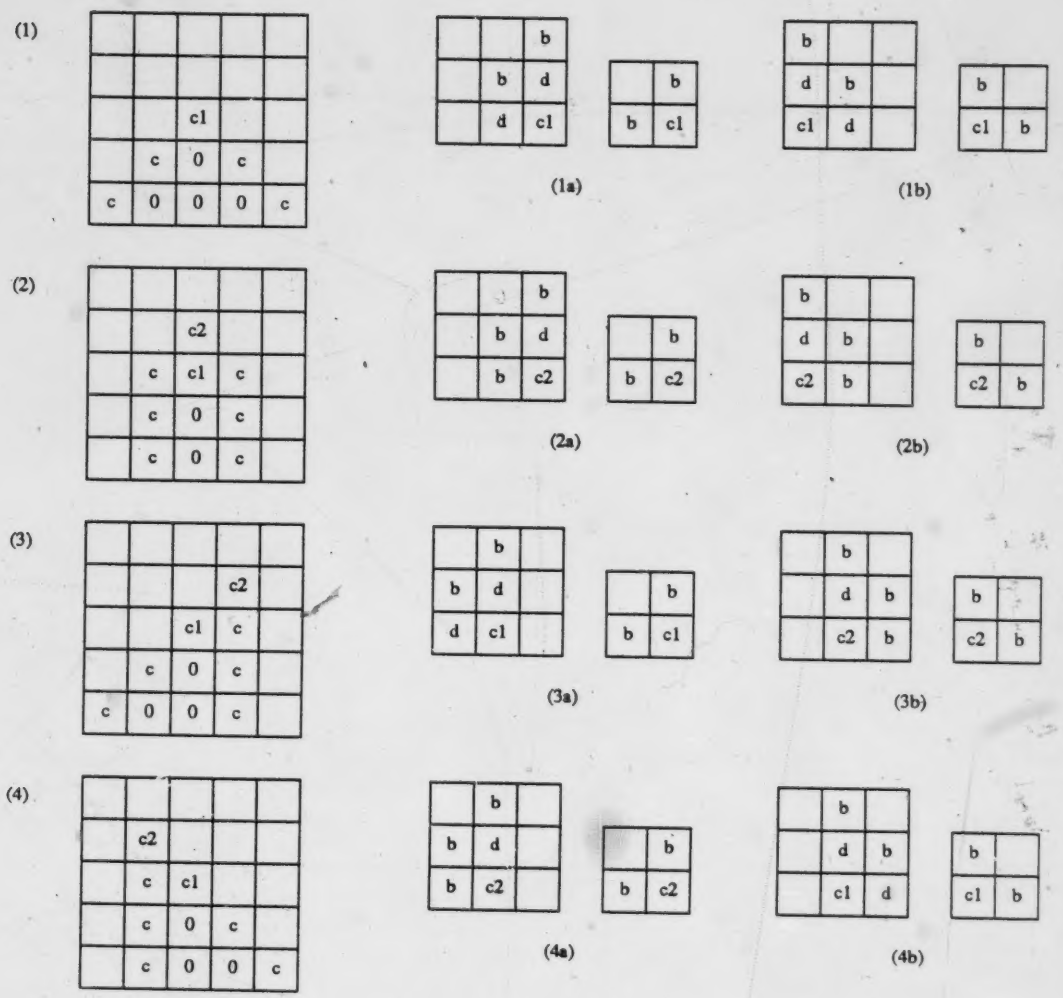

Figure 4.4 Window Patterns for Pcode 13

The following two examples show the relation between the pcode of a c-pixel and its window pattern. First, assume that a c-pixel has the pcode 01; the corresponding 3×3 window is the first window in Fig. 4.2; if this window is matched or all pixels in the window are black, its window pattern '01' (Fig. 4.3) is copied. Second, the pcode 13 of a c-pixel applies two window

patterns at a time, according to the 5×5 window. If the window 3 is matched, these two patterns, 3a and 3b with c1 and c2 as the c-pixels, are applied. The two 3×3 windows are the 180° rotations of the second window in Fig. 4.2, and the 2×2 windows for 3a and 3b are, respectively, the 180° and 270° rotations of the third window.

4.2. New Thinning Algorithm

This algorithm is quite different from the existing algorithms and does not apply its deletion rules as long as the 3×3 or 2×2 windows can be used. The algorithm is as follows.

```
do { /* iteratively */
     1.a. find contour pixels and mark by 'c' (c-pixels)
       b. find pcodes for all c-pixels
     2. for every c-pixel with its pcode,
          apply 3×3 window
          if the window is matched, /* all pixels in the window are black */
               then {
                    copy the corresponding 3×3 window pattern
                    change c-pixels in the window into d-pixels  /* change 'c' to 'd' */
                    }
               else {
                    apply 2×2 window
                    if the window is matched,
                         then {
                              copy the corresponding 2×2 window pattern
                              change c-pixels in the window into d-pixels
                              }
                    else if the c-pixel is black,
                         then apply the deletion rules   /* table reformulation */
                    }
     3.delete all d-pixels
} until (no more pixel deletion)
```

The operation used by this algorithm is *contour-tracing*. In the first step, contour pixels and

their pcodes are determined. In the second step, deletable pixels are marked by 'd' or directly deleted, depending on the windows applied. According to its pcode, every c-pixel is examined by applying 3×3 or 2×2 windows or deletion rules in the reformulated table (Chap. 3, Table 3.4). At first, a 3×3 window is applied. If the window is matched, the corresponding pattern is copied and the labels of c-pixels in the window are changed to 'd'; otherwise, the c-pixel is examined by applying a 2×2 window. Similarly, if the 2×2 window is matched, the 2×2 window pattern is copied and the labels of c-pixels in the window are changed to 'd'; otherwise, the deletion rules in Section 3.4 are used to examine the c-pixel whether or not the pixel is deleted. The third step deletes all pixels 'd'. The three steps are implemented iteratively until no more pixel is deleted.

4.2.1. Satisfying Criteria

The basic idea of the algorithm is to determine next contour pixels iteratively applying the windows and window patterns. Starting from the outermost layer of pixels, the process is performed along the object contour to incorporate the criteria to generate desirable skeletons.

The window patterns of adjacent c-pixels are overlapped so that next contour pixels are connected as well as the previous ones, and thus connectivity and the shape of object are preserved. Started from the outermost layer of pixels, an alternate pixel deletion, i.e. outer and inner thinning process, may result in a skeleton in the center of the original object. This process will also produce 1-pixel width, no endpoint shrinkage, and no noisy-branch skeletons. However, in the case of noise in an object, a noisy branch may occur. In general, the algorithm can generate skeletons which preserve the global properties of the object.

4.2.2. Sensitivity

Skeletons generated are usually affected by noise, starting point and direction of pixel deletion. The sensitivity of the new algorithm is described.

In the first figure of Fig. 4.5, pixel c3 is considered as noise. The pcode of this pixel is 03, which does not have any corresponding window. This pixel is deletable according to deletion rules. However, this influences the next deleting process, i.e. examining pixel c4. The pcode of c4 is 32 and based on the 2×2 window pattern, this pixel will be the next contour pixel. A noisy branch will appear. If pixel c3 does not exist, the pcode of c4 is 22 and according to its 3×3 window pattern, pixel c2 together with c1 and c4 are deleted and the noisy branch will not occur.

The second figure of Fig. 4.5 shows that pixel c3 is noise. This pixel and pixel c2 are not deletable and result in a noisy branch. If the noise is eliminated, pixel c2 will also be deleted. In general, the algorithm is sensitive to noise.

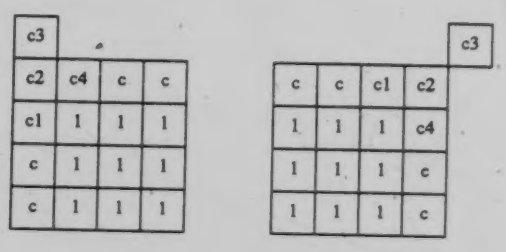

Figure 4.5 Noise

Because of its window usage and simultaneously pixel deletion in every iteration, the algorithm is not sensitive to starting point position and direction of the deletion process. According to the pcodes of c-pixels, windows are applied along the contour which may be in a clockwise or counter clockwise direction, and the window patterns determine the pixels of the next contour, and the pixels to be deleted. This process may be started from any c-pixel with any direction without affecting the results. The skeletons of an object will be the same. However, if the width of the character is an even number, for example two or four pixels, the positions of the skeleton will be different, depending on the starting points and directions of the thinning process.

4.2.3. Time Complexity

The time complexity is measured based on the number of time units used by every step in the algorithm. In Step 1, determining a c-pixel and its pcode needs at most 8 comparisons. In Step 2, there are three possible cases for every c-pixel. First, matching a 3×3 window needs at most 9 comparisons, and copying the 3×3 window pattern needs 8 time units; second, matching a 2×2 window needs at most 4 comparisons, and copying the 2×2 window pattern needs 3 units; and third, applying the deletion rules needs at most 16 units, and 4 units for searching an entry in the table containing 15 entries. In Step 3, deleting a d-pixel needs 1 unit, but the whole image should be scanned. Therefore, the processing time depends on the number of c-pixels, the number of skeletal pixels, and the number of the pixels in the image. Table 4.1 presents the components of the processing time. The time complexity mainly depends on the size of the image, i.e. $O(n^2)$.

Table 4.1 Components of Processing Time

Step Number	Components	Time Units per Pixel
1	determining a c-pixel and pcode	8
2	matching a 3×3 window	9
	copying a 3×3 window pattern	8
	matching a 2×2 window	4
	copying a 2×2 window pattern	3
	evaluating deletion rules	20
3	deleting a d-pixel	1

4.3. Results and Comparisons

Some skeletons generated by the algorithm are presented in Fig. 4.6, and compared to those generated by existing algorithms (Appendix A2). Criteria satisfied, time and storage requirement are also compared.

4.3.1. Skeletons and Criteria Satisfied

The skeletons produced by the new algorithm satisfy the criteria, i.e. 8-connected, 1-pixel wide, shape adaptive, in the center of the original object, no noisy branch, and no endpoint shrinkage, except the skeletons of 'Q' and 'R' have some distortions. However, these skeletons are better than those in Appendix A2, especially the skeletons of 'K', 'M' and 'N'. The skeletons of characters 'A', 'O', 'Q' and 'V' produced by the algorithm G2 are improved.

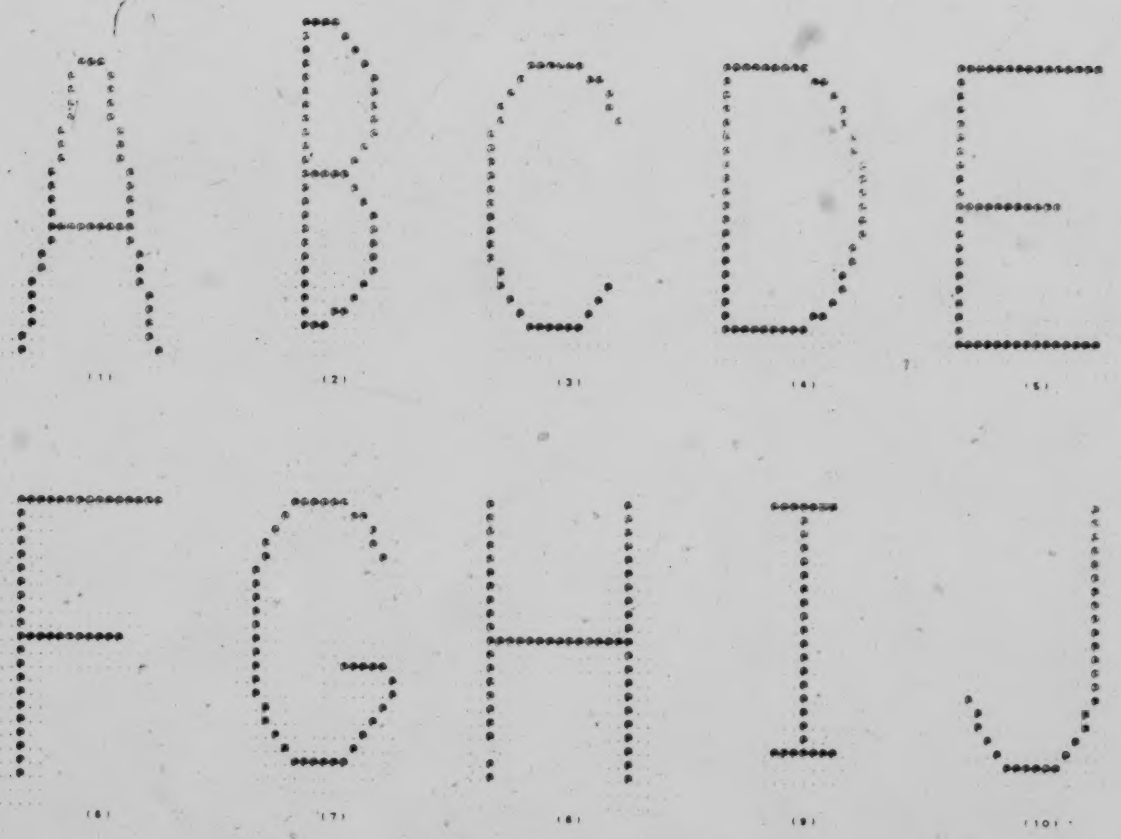

Figure 4.6 Skeletons generated by the New Algorithm

Figure 4.6. (cont.)

Figure 4.6. (cont.)

4.3.2. Time and Storage Requirement

The time and storage requirement of the new algorithm is compared to that of the algorithms CH, T2 and G2 (Chap. 3, Table 3.6 and 3.7).

Assume that the time for a thinning process is measured in terms of the number of 1-pixels in the object (b), and consider the time units in Table 4.1. The time used for contour tracing is $8b$ units and for matching 3×3 windows is $9b$ units. If ¾ of the number of object pixels matches the 3×3 windows, the time for copying a 3×3 window is $6b$ units. If ½ of the number of object pixels are examined by 2×2 windows, the time for matching these windows is $2b$ units. If ¼ of the number of object pixels matches the 2×2 windows, the time for copying 2×2 window patterns is $\frac{1}{4}b$ units. If the skeletal pixels are one-third of the number of object pixels, the time for evaluating the deletion rules is $7b$ units. Deleting d-pixels needs n^2 units in an iteration. If $n^2=2b$, the time used to delete d-pixels in 3 iterations is 6b units. The total time is about $39b$ units.

The original algorithm G2 requires 14c units and the reformed algorithm G2 requires 16c units in each iteration (Chap. 3, Table 3.7). If the number of contour pixels in the whole process is assumed to be the same as the number of object pixels, i.e. c=b, the first algorithm needs 14b units and the second algorithm needs 16b units. The time units required will be 22b and 24b units, if the time for contour tracing (8b units) is included. If searching the table containing 15 entries needs 4 time units per pixel, the reformed G2 requires 28b units. Therefore, the time of the new algorithm is more than that of the algorithms G2, either original or reformed algorithm (Table 4.2). The time is less than that of the reformed CH and T2 (Chap. 3, Table 3.7) because the two algorithms apply 4-subiterative operations. However, the execution time of the new algorithm is about 2.65 seconds which is more than that of the other algorithms (Chap. 2, Table 2.3). In general, the new algorithm is slower than the existing ones.

Because the algorithm uses the table containing 15 entries, the storage requirement is the same as that of the reformed CH and DE, but more than that of the reformed ZS, T1, T2, G1 and

G2 and less than that of the reformed NS and PA (Chap. 3, Table 3.5).

Table 4.2 Time and Storage Requirement of New Algorithm and G2

Algorithm	Total Time (units)	Exec.Time (seconds)	Storage (bytes)
New Algorithm	39b	2.65	15
Original G2	22b	1.14	256
Reformed G2	28b	1.39	10

4.4. Summary

The new thinning algorithm applies some 3×3 and 2×2 windows, and uses 15 table entries as the deletion rules if no window is matched. This algorithm requires the same storage as the reformed CH and DE, and generates better skeletons but needs more processing time than the existing algorithms.

Chapter 5

Conclusion and Future Work

5.1. Conclusions

In this thesis, we reviewed nine well-known thinning algorithms with experiments on six criteria, investigated the table lookup approach and the table reformulation, and proposed a new thinning algorithm. The algorithm comparison is based on skeletons, and time and storage requirement.

A desirable skeleton should be 8-connected, 1-pixel wide, in the center of original object, stable, shape adaptive and without noisy branches. The quality of a skeleton depends on the deletion rules and operations used by an algorithm, noise, and starting points and directions of the deleting process. Most skeletons of the objects in the experiments satisfy the criteria.

The algorithms with contour-tracing operations are faster than those with subiterative operations. The 2-subiterative algorithms are faster than the 4-subiterative ones. However, all algorithms are $O(n^2)$, where n is the size of an image.

The algorithms with table lookup are faster than those with direct computation, but require some storage. The storage requirement in an algorithm with contour-tracing is less than that in a subiterative algorithm. A 2-subiterative algorithm needs less storage than a 4-subiterative algorithm. By reformulating the lookup table, the storage requirement is reduced, but the searching time is longer than that used to access the original tables. However, the reformulated table is flexible and thus useful as an experimental tool.

The new thinning algorithm can generate better skeletons but requires more processing time than existing algorithms. This algorithm is designed for English characters, but may not work for

other line like objects.

5.2. Further Work

Skeletons are compared visually based on the criteria. No quantitative measurement can be used to compare skeletons. If a quantitative criterion exists, skeleton comparison will be easier and human independent.

The main problem of the thinning process is to eliminate noise and to incorporate the shape as a global property of an object. Thinning algorithms, which examine image pixels one by one and only use local information, should be able to remove noise such that the shape of the object can be preserved and smoothing is not required. The algorithms should have the property of not being sensitive to noise.

This is similar to human thinning. In the human thinning process, a global view of the entire object (character) often affects skeleton generation. The person is doing thinning and recognizing the character at the same time and uses a great deal of knowledge in the thinning process. If human thinning can be simulated by a thinning algorithm in some way, better results may be expected.

The requirement of less time is another issue in the thinning process. Besides achieving better skeletons and less storage, a thinning algorithm should be as efficient as possible.

References

[Arc79] C. Arcelli, A Condition for Digital Points Removal, *Signal Processing 1*, (1979), 283-285.

[ChH88] Y. S. Chen and W. H. Hsu, A Modified Fast Parallel Thinning Algorithm for Digital Patterns, *Pattern Recognition Letters 7*, 2 (February 1988), 99-106 .

[ChS86] Y. K. Chu and C. Y. Suen, An Alternative Smoothing and Stripping Algorithm for Thinning Digital Binary Patterns, *Signal Processing 11*, (1986), 207-222.

[CZS86] N. Chuei, T. Y. Zhang and C. Y. Suen, New Algorithms for Thinning Binary Images and Chinese Characters, *Computer Processing of Chinese & Oriental Languages 2*, 3 (May 1986), 169-179.

[DaP81] E. R. Davies and A. P. N. Plummer, Thinning Algorithms: A Critique and A New Methodology, *Pattern Recognition 14*, 1-6 (1981), 53-63.

[Deu72] E. S. Deutsch, Thinning Algorithms on Rectangular, Hexagonal, and Triangular Arrays, *Communications of the ACM 15*, 9 (September 1972), 827-837.

[Fre61] H. Freeman, On The Coding of Arbitrary Geometric Configurations, *IRE Trans. Electronic Computer EC-10*, (1961), 260-268.

[GoS87] V. K. Govindan and A. P. Shivaprasad, A Pattern Adaptive Thinning Algorithm, *Pattern Recognition 20*, 6 (1987), 623-637.

[GuH89] Z. Guo and R.W. Hall, Parallel Thinning with Two-Subiteration Algorithms, *Communications of the ACM 32*, 3 (March 1989), 359-373.

[Hal89] R.W. Hall, Fast Parallel Thinning Algorithms: Parallel Speed and Connetivity Preservation, *Communications of the ACM 32*, 1 (January 1989), 124-131.

[Hil69] C. J. Hilditch, Linear Skeletons from Square Cupboards, in *Machine Intelligence*, vol. 4, Meltzer and Michie (ed.), New York, 1969, 403-420.

[HSC87] C. M. Holt, A. Stewart, M. Clint and R. H. Perrott, An Improved Parallel Thinning Algorithm, *Communications of the ACM 30*, 2 (February 1987), 156-160.

[Kwo88] P.C.K. Kwok, A Thinning Algorithm by Contour Generation, *Communications of the ACM 31*, 11 (November 1988), 1314-1324.

[Lu85] H. E. Lu and P. S. P. Wang, An Improved Fast Parallel Thinning Algorithm for Digital Patterns, *IEEE Proc. of Computer Vision and Pattern Recognition*, 1985, 364-367.

[NaS84a] N. J. Naccache and R. Shinghal, SPTA: A Proposed Algorithm for Thinning Binary Patterns, *IEEE Trans. on Systems, Man, and Cybernetics SMC-14*, 3 (May/June 1984), 409-418.

[NaS84b] N. J. Naccache and R. Shinghal, An Investigation into the Skeletonization Approach of Hilditch, *Pattern Recognition 17*, 3 (1984), 279-284.

[Pav81] T. Pavlidis, A Flexible Parallel Thinning Algorithm, *IEEE Int. Conf. on Pattern Recognition and Image Processing*, August 1981, 162-167.

[Pav82] T. Pavlidis, *Algorithms for Garphics and Image Processing*, Computer Press, New York, 1982.

[PlS88] R. Plamondon and C.Y. Suen, On The Definition of Reference Skeletons for Comparing Thinning Algorithms, *Proc. Vision Interface '88*, Edmonton, Alberta, June 1988, 70-75.

[RoP66] A. Rosenfeld and J.L. Pfaltz, Sequential Operations in Digital Picture Processing, *J. ACM 13*, (1966), 471-494.

[Ros75] A. Rosenfeld, A Characterization of Parallel Thinning Algorithms, *Information and Control 29*, (1975), 286-291.

[Rut66] D. Rutovitz, Pattern Recognition, *J. Royal Statist. Soc. 129*, (1966), 504-530.

[Smi87] R. W. Smith, Computer Processing of Line Images: A Survey, *Pattern Recognition 20*, 1 (1987), 7-15.

[Sob78] I. Sobel, Neighborhood Coding of Binary Images for Fast Contour Following and General Binary Array Processing, *Computer Graphics and Image Processing*, 1978, 127-135.

[StR71] R. Stefanelli and A. Rosenfeld, Some Parallel Thinning Algorithms for Digital Pictures, *Journal of the ACM 18*, 2 (April 1971), 255-264.

[SuA87] Satoshi Suzuki and Keiichi Abe, Binary Picture Thinning by An Iterative Parallel Two-Subycle Operation, *Pattern Recognition 20*, 3 (1987), 297-307.

[Tam78] H. Tamura, A Comparison of Line Thinning Algorithms from Digital Geometry Viewpoint, *Proc. 4th Int. Conf. on Pattern Recognition*, 1978, 715-719.

[WHF87] P. S. P. Wang, Lai-Wah Hui and Thomas Fleming Jr., Further Improved Fast Parallel Thinning Algorithm for Digital Patterns, *IEEE Int. Conf. on Pattern Recognition*, 1987, 37-40 .

[ZhS84] T. Y. Zhang and C. Y. Suen, A Fast Parallel Algorithm for Thinning Digital Patterns, *Communications of the ACM 27*, 3 (March 1984), 236-239.

Appendix A1

Rule Windows

```
011 001 011 000 001 011 000 000 001 011
0p0 0p1 0p1 0p1 0p1 0p1 0p0 0p1 0p1 0p1
000 000 000 001 001 001 011 011 011 011

000 000 000 001 011 000 000 000 000 001
0p0 0p0 0p1 0p1 0p1 1p0 1p0 1p0 1p1 1p1
110 111 111 111 111 100 110 111 111 111

110 111 111 111 111 100 110 111 100 110
0p0 0p0 0p1 0p1 0p1 1p0 1p0 1p0 1p0 1p0
000 000 000 001 011 000 000 000 100 100

111 100 100 100
1p0 1p0 1p0 1p1
100 110 111 111
```

Figure A1.1 Rule Windows of ZS (N-W subiteration)

```
000 011 001 011 000 001 011 000 000 001
0p0 0p0 0p1 0p1 0p1 0p1 0p1 0p0 0p1 0p1
000 000 000 000 001 001 001 011 011 011

011 000 000 000 001 011 011 000 000 010
0p1 0p0 0p0 0p1 0p1 0p1 1p0 1p0 1p0 1p0
011 110 111 111 111 111 000 011 100 100

011 000 000 110 111 111 111 111 111 100
1p0 1p0 1p0 0p0 0p0 0p1 0p1 0p1 0p1 1p0
100 110 111 000 000 000 001 011 111 000

110 111 111 111 100 100 100 110 111 111
1p0 1p0 1p1 1p1 1p0 1p0 1p0 1p0 1p0 1p1
000 000 000 001 010 011 100 100 100 100

111 100 100
1p1 1p0 1p0
101 110 111
```

Figure A1.2 Rule Windows of DE (S-W subiteration)

```
011 010 011 010 011 000 001 000 000 001
0p0 0p1 0p1 0p1 0p1 0p1 0p1 0p0 0p1 0p1
000 000 000 001 001 010 010 011 011 011

000 000 001 000 000 001 010 011 010 011
0p0 0p1 0p1 0p0 0p1 0p1 1p0 1p0 1p1 1p1
110 110 110 111 111 111 000 000 000 000

010 011 000 000 001 000 000 001 010 011
1p1 1p1 1p0 1p1 1p1 1p0 1p1 1p1 1p0 1p0
001 001 010 010 010 011 011 011 100 100

010 011 010 011 000 000 001 000 000 001
1p1 1p1 1p1 1p1 1p0 1p1 1p1 1p0 1p1 1p1
100 100 101 101 110 110 110 111 111 111

110 111 110 111 110 111 110 111 110 111
0p0 0p0 0p1 0p1 0p1 0p1 1p0 1p0 1p1 1p1
000 000 000 000 001 001 000 000 000 000

110 111 100 100 101 100 100 101 110 111
1p1 1p1 1p0 1p1 1p1 1p0 1p1 1p1 1p0 1p0
001 001 010 010 010 011 011 011 100 100

110 111 110 111 100 100 101 100 100 101
1p1 1p1 1p1 1p1 1p0 1p1 1p1 1p0 1p1 1p1
100 100 101 101 110 110 110 111 111 111
```

Figure A1.3 Rule Windows of NS (N-S subiteration)

```
011 001 011 000 001 011 111 111 111 111
0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
001 011 011 111 111 111 000 001 011 111
```

Figure A1.4 Rule Windows of T1 (W subiteration)

```
000  111  110  111  100  110  111  100  110  111
1p0  1p0  1p0  1p0  1p0  1p0  1p0  1p0  1p0  1p0
111  000  100  100  110  110  110  111  111  111
```

Figure A1.5 Rule Windows of T1 (E subiteration)

```
001  000  001  000  000  001  100  100  100  101
0p1  0p1  0p1  1p0  1p1  1p1  1p0  1p0  1p1  1p1
011  111  111  111  111  111  110  111  111  111
```

Figure A1.6 Rule Windows of T1 (N subiteration)

```
011  001  011  000  001  011  000  000  001  011
0p0  0p1  0p1  0p1  0p1  0p1  0p0  0p1  0p1  0p1
000  000  000  001  001  001  011  011  011  011

000  000  000  001  011  110  111  111  111  111
0p0  0p0  0p1  0p1  0p1  0p0  0p0  0p1  0p1  0p1
110  111  111  111  111  000  000  000  001  011

111
0p1
111
```

Figure A1.7 Rule Windows of T2 (W subiteration)

```
011  000  000  000  000  000  000  110  111  100
0p0  0p0  0p0  0p0  1p0  1p0  1p0  0p0  0p0  1p0
000  011  110  111  100  110  111  000  000  000

110  111  100  110  111  100  110  111  100  110
1p0  1p0  1p0  1p0  1p0  1p0  1p0  1p0  1p0  1p0
000  000  100  100  100  110  110  110  111  111

111
1p0
111
```

Figure A1.8 Rule Windows of T2 (E subiteration)

```
001  000  001  000  000  001  000  000  000  001
0p1  0p1  0p1  0p0  0p1  0p1  0p0  0p0  0p1  0p1
000  001  001  011  011  011  110  111  111  111

000  000  000  000  001  100  100  100  100  100
1p0  1p0  1p0  1p1  1p1  1p0  1p0  1p0  1p0  1p1
100  110  111  111  111  000  100  110  111  111

101
1p1
111
```

Figure A1.9 Rule Windows of T2 (N subiteration)

```
011 010 001 011 000 010 001 011 000 010
0p0 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
000 000 000 000 001 001 001 001 010 010

001 011 000 000 010 001 011 000 000 010
0p1 0p1 0p0 0p1 0p1 0p1 0p1 0p0 0p1 0p1
010 010 011 011 011 011 011 110 110 110

001 011 000 000 010 001 011 010 011 010
0p1 0p1 0p0 0p1 0p1 0p1 0p1 1p0 1p0 1p1
110 110 111 111 111 111 111 000 000 000

011 010 011 000 010 011 000 001 000 010
1p1 1p1 1p1 1p0 1p0 1p0 1p1 1p1 1p0 1p0
000 001 001 010 010 010 010 010 011 011

011 000 001 000 010 011 010 011 010 011
1p0 1p1 1p1 1p0 1p0 1p0 1p1 1p1 1p1 1p1
011 011 011 100 100 100 100 100 101 101

000 010 011 000 001 000 010 011 000 001
1p0 1p0 1p0 1p1 1p1 1p0 1p0 1p0 1p1 1p1
110 110 110 110 110 111 111 111 111 111

110 111 110 111 110 111 110 111 110 111
0p0 0p0 0p1 0p1 0p1 0p1 0p1 0p1 0p1 0p1
000 000 000 000 001 001 010 010 011 011

110 111 110 111 100 110 111 110 111 110
0p1 0p1 0p1 0p1 1p0 1p0 1p0 1p1 1p1 1p1
110 110 111 111 000 000 000 000 000 001

111 100 110 111 100 101 100 110 111 100
1p1 1p0 1p0 1p0 1p1 1p1 1p0 1p0 1p0 1p1
001 010 010 010 010 010 011 011 011 011

101 100 110 111 110 111 110 111 100 110
1p1 1p0 1p0 1p0 1p1 1p1 1p1 1p1 1p0 1p0
011 100 100 100 100 100 101 101 110 110

111 100 101 100 110 111 100 101
1p0 1p1 1p1 1p0 1p0 1p0 1p1 1p1
110 110 110 111 111 111 111 111
```

Figure A1.10 Rule Windows of PA

Appendix A2

Results of Existing Thinning Algorithms

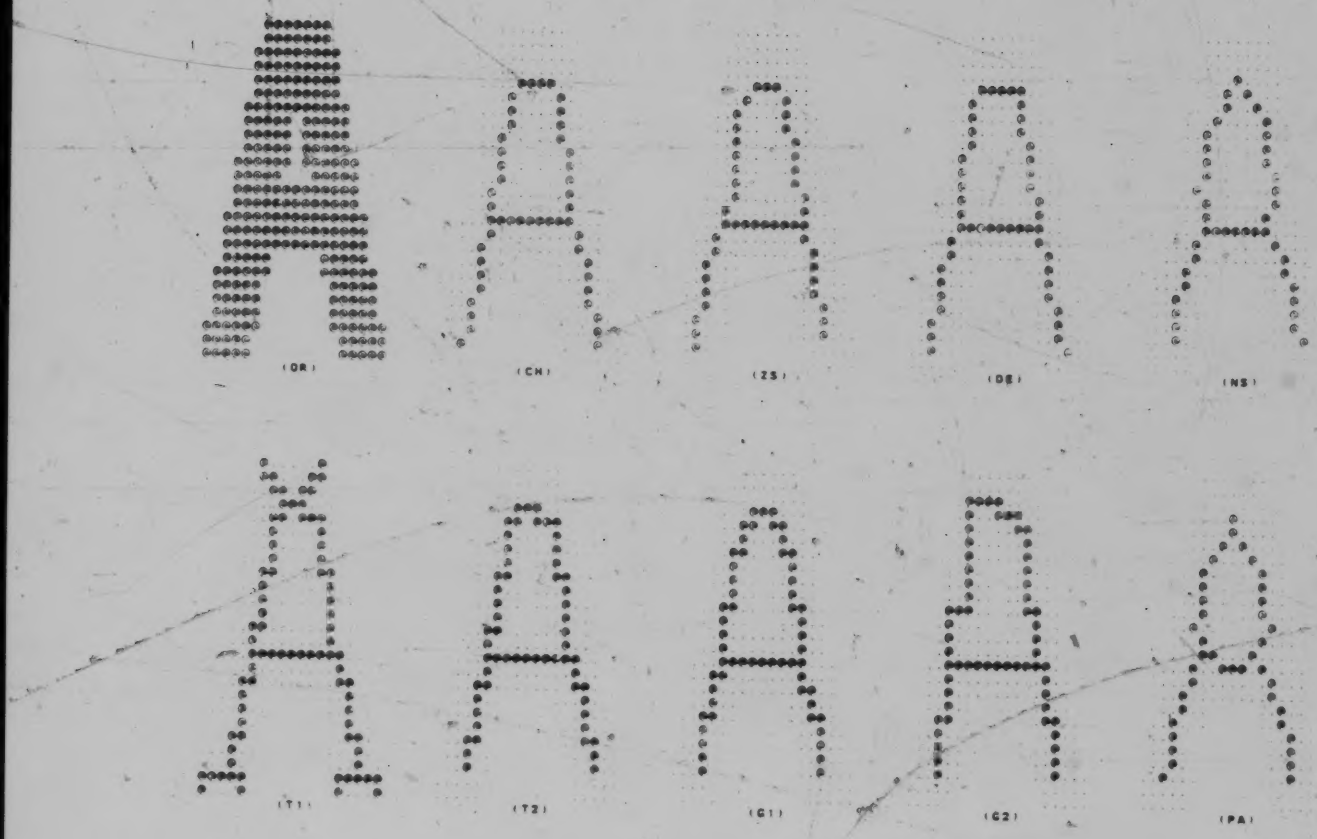

Figure A2.1 Skeletons of 'A'

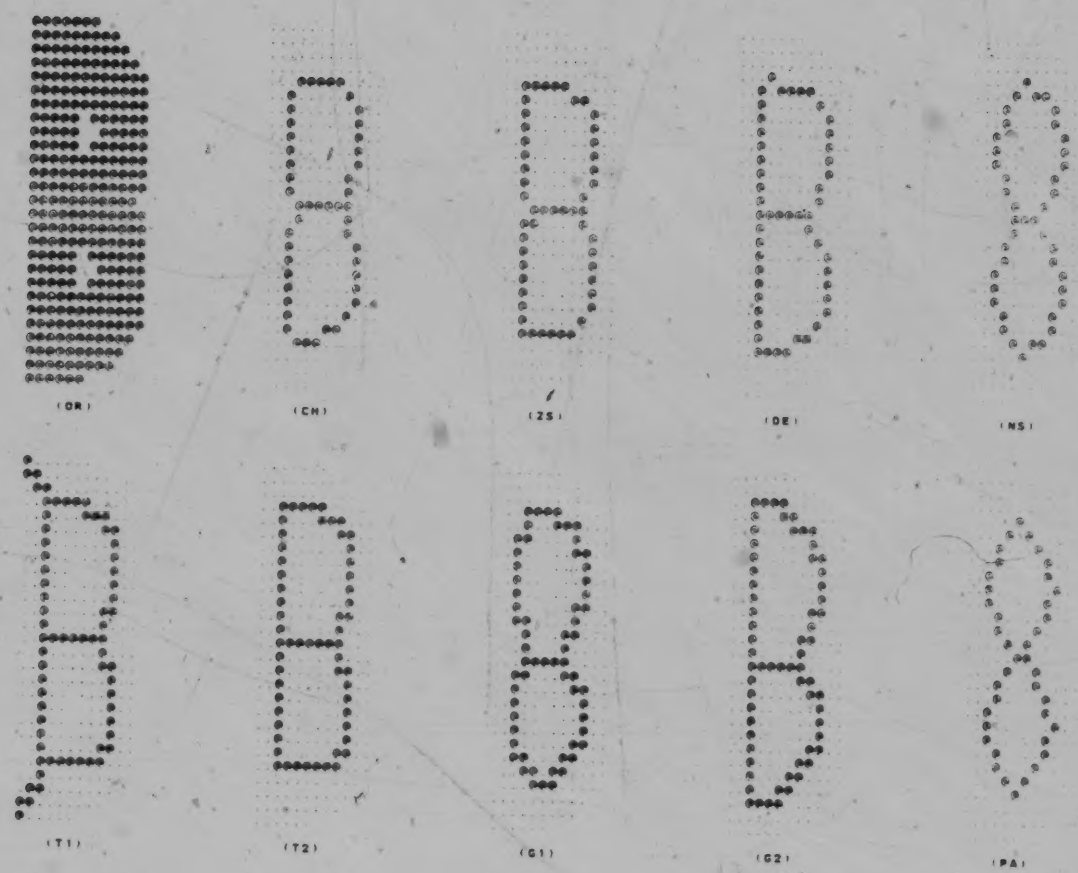

Figure A2.2 Skeletons of 'B'

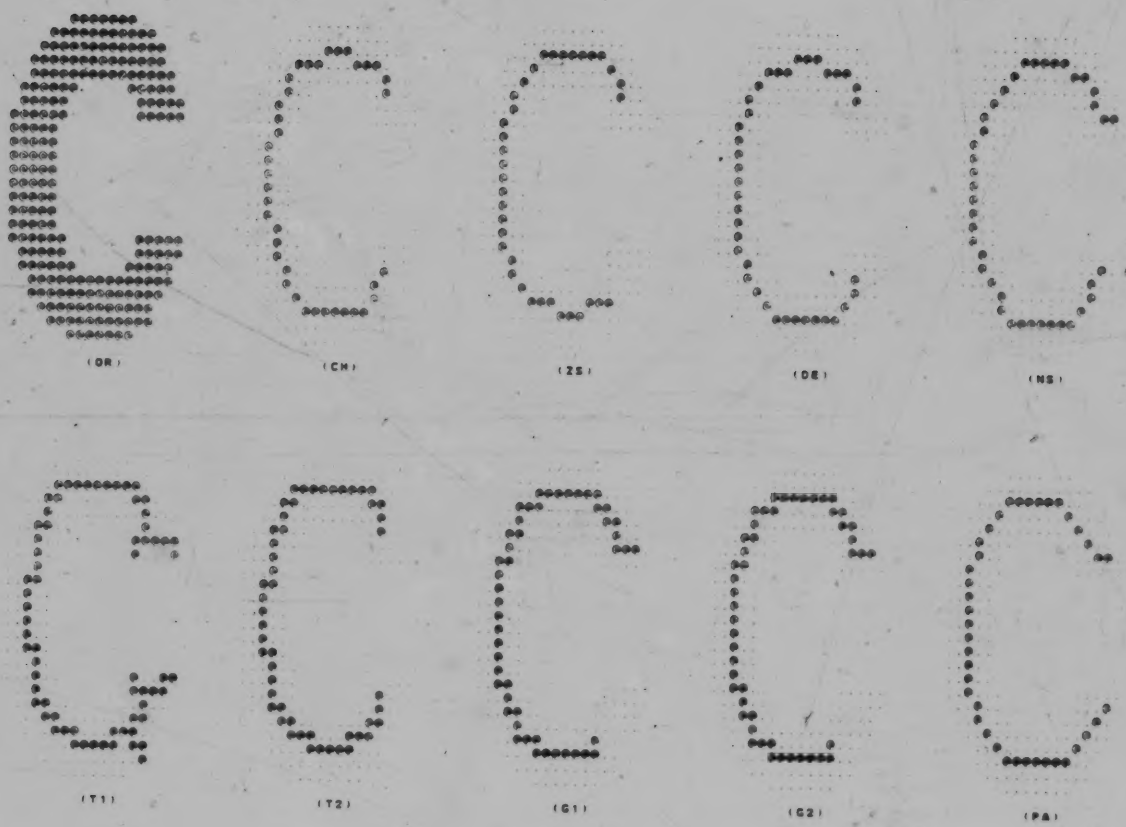

Figure A2.3 Skeletons of 'C'

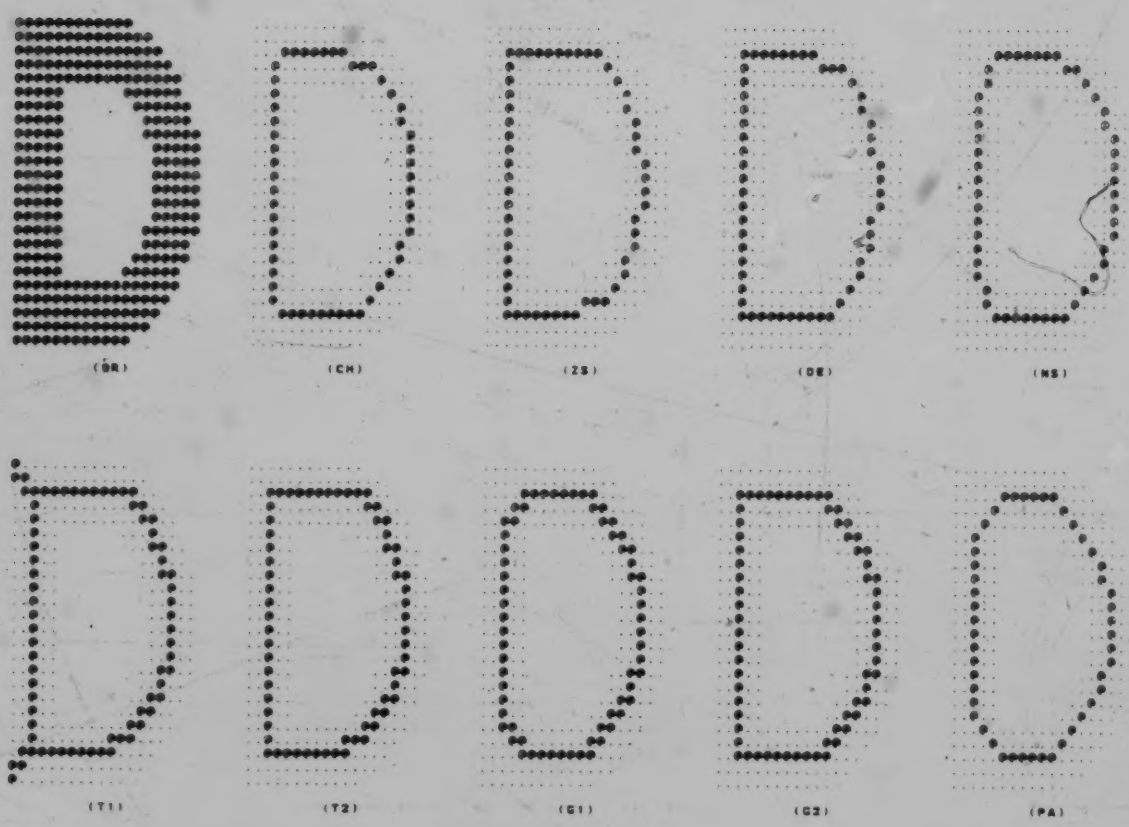

Figure A2.4 Skeletons of 'D'

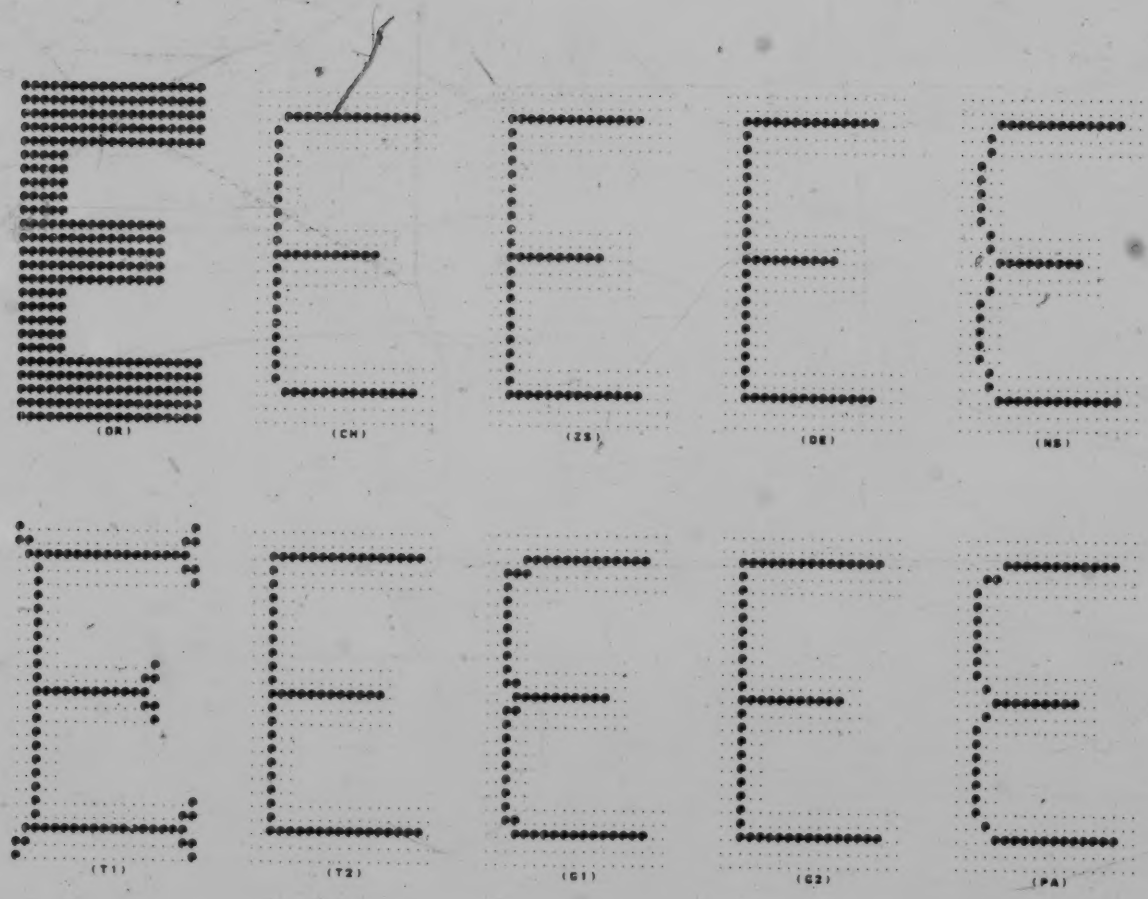

Figure A2.5 Skeletons of 'E'

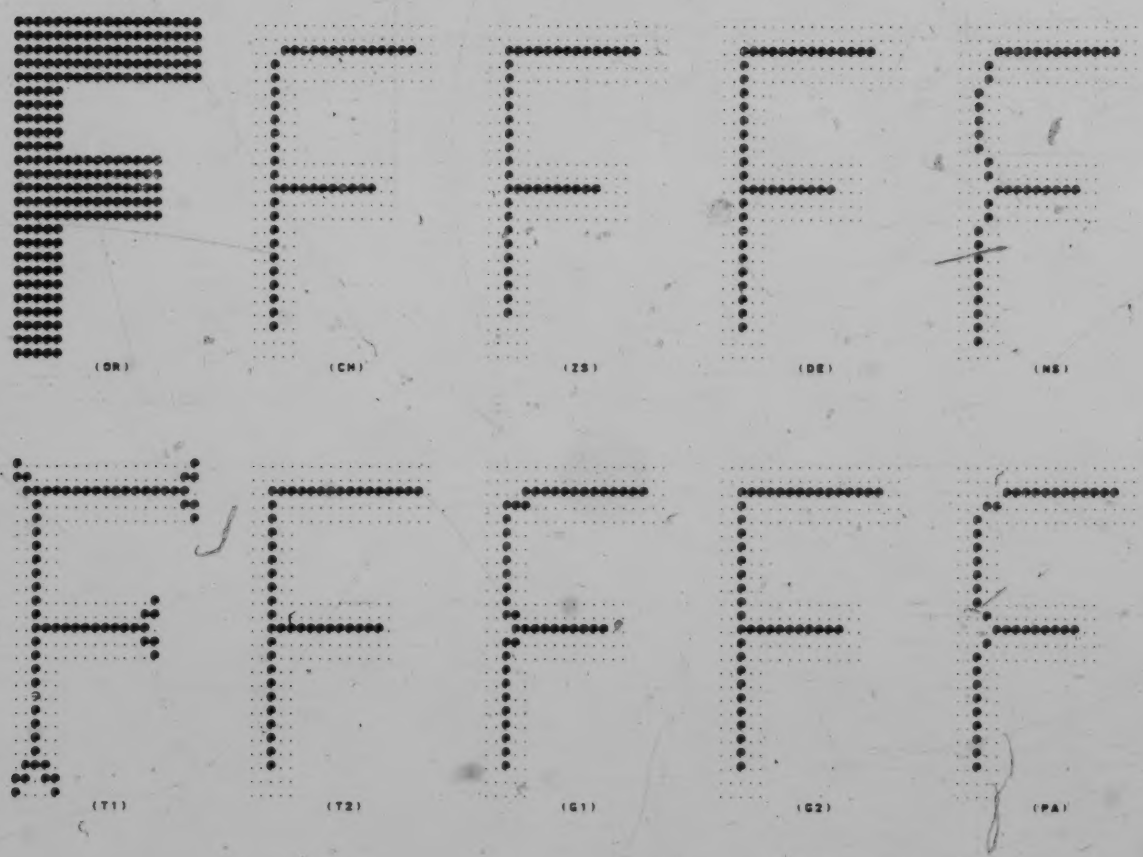

Figure A2.6 Skeletons of 'F'

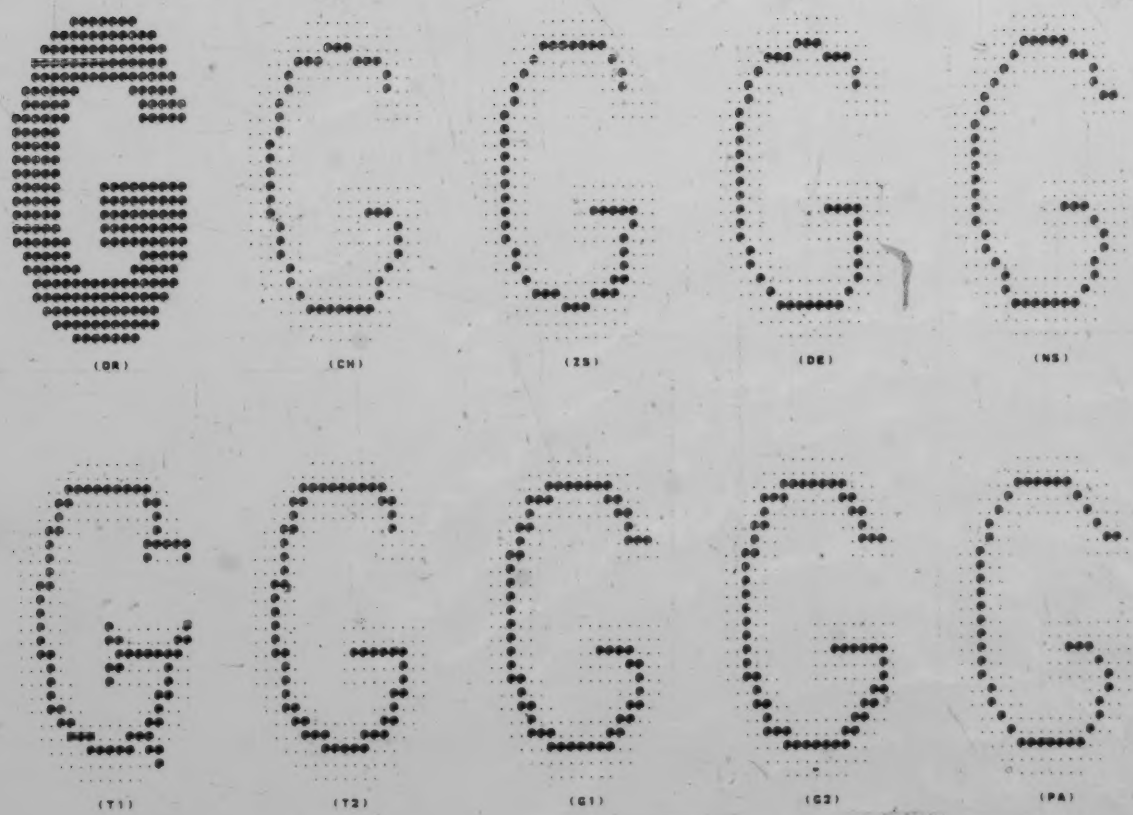

Figure A2.7 Skeletons of 'G'

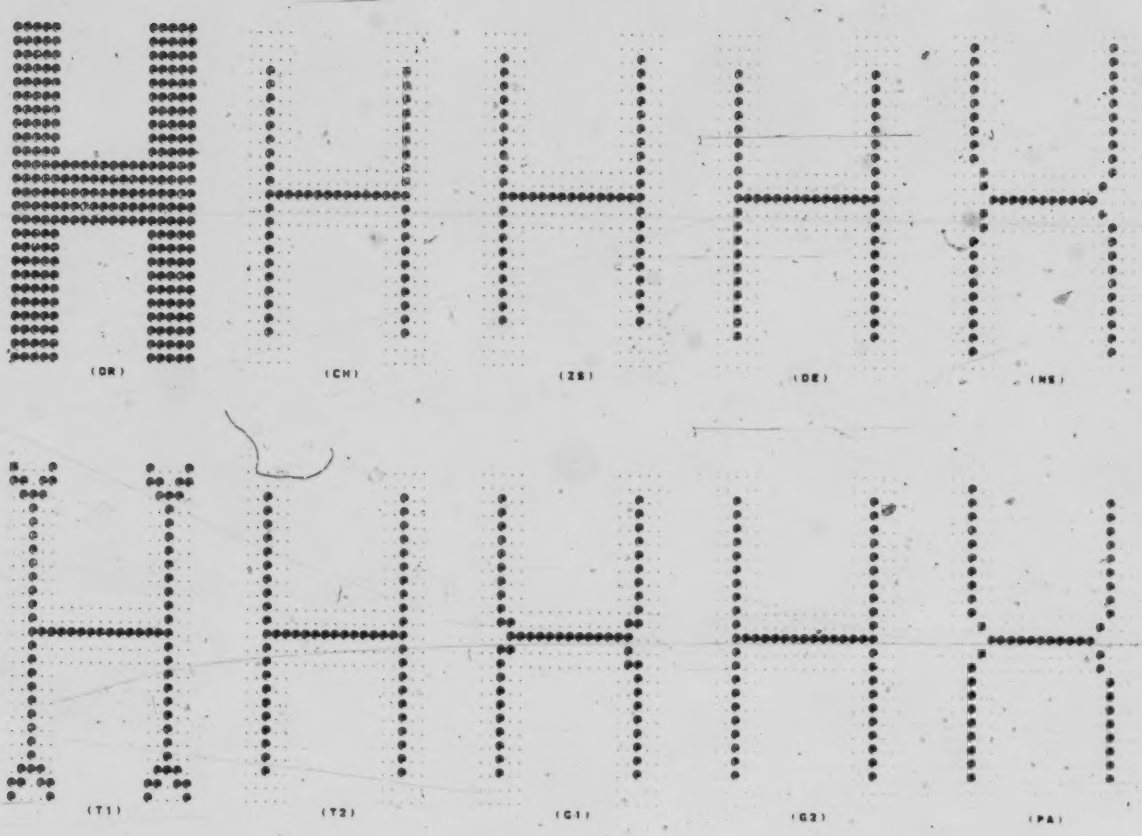

Figure A2.8 Skeletons of 'H'

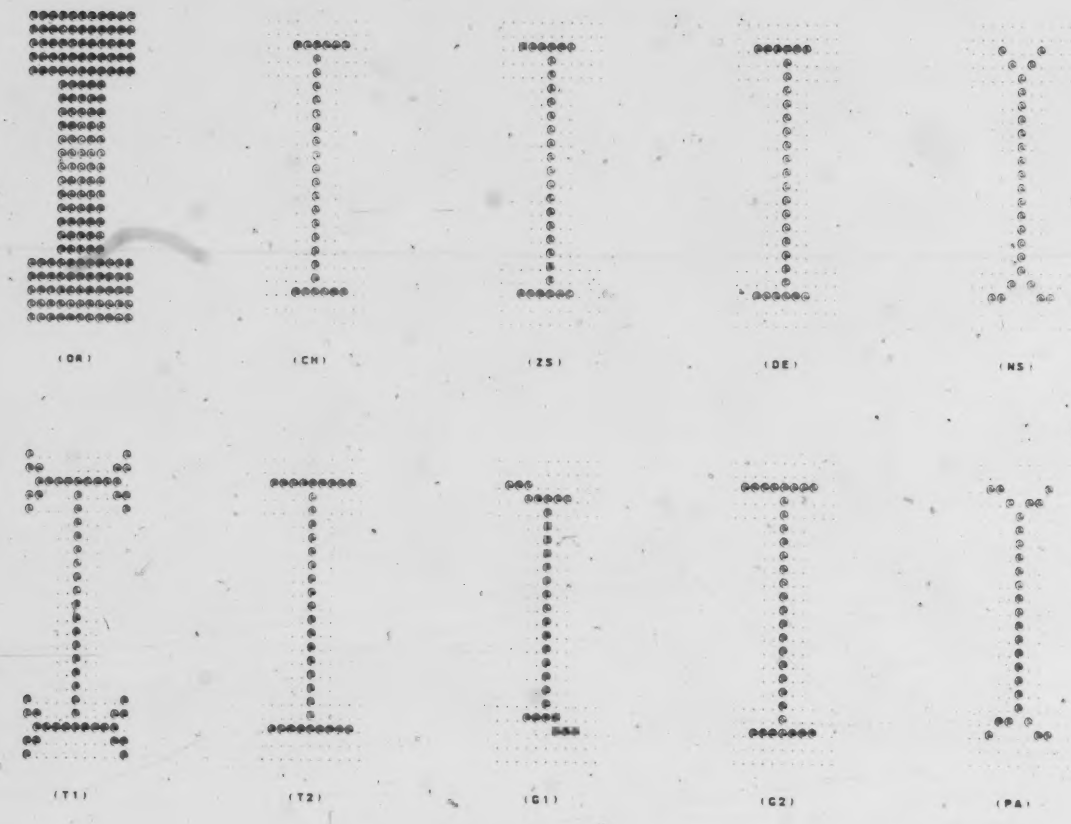

Figure A2.9 Skeletons of 'I'

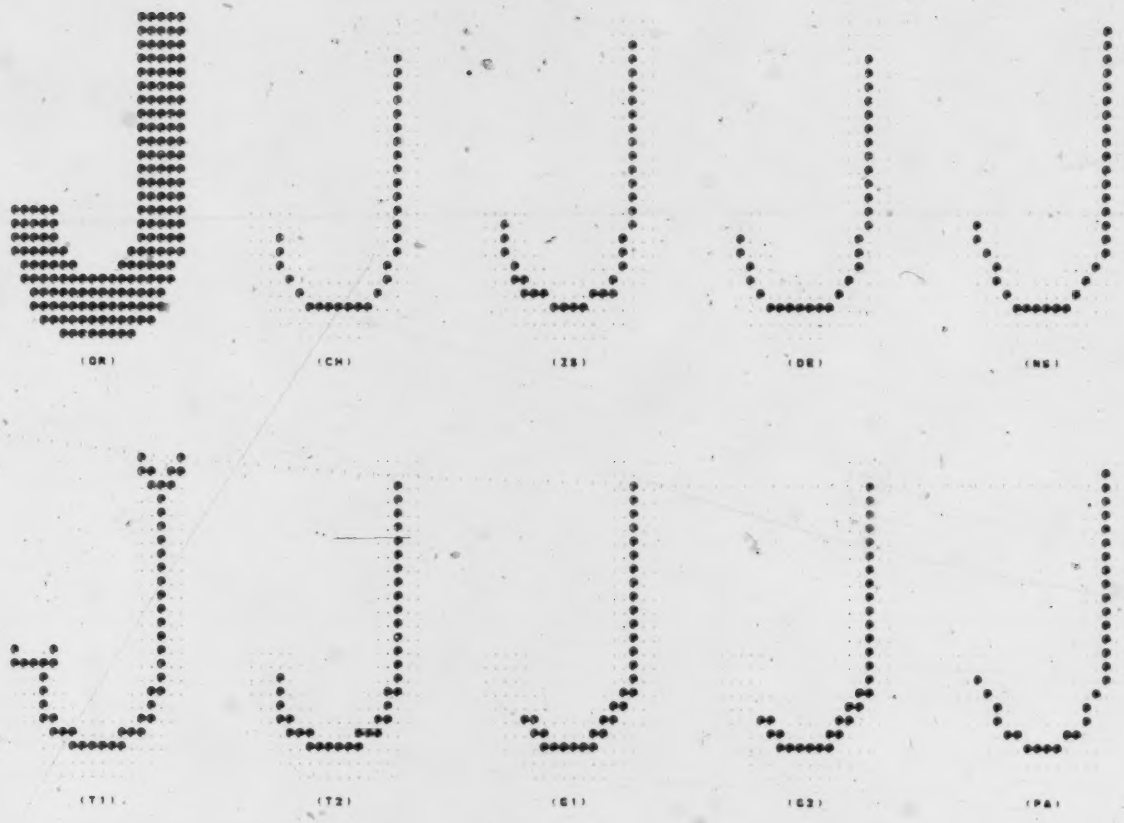

Figure A2.10 Skeletons of 'J'

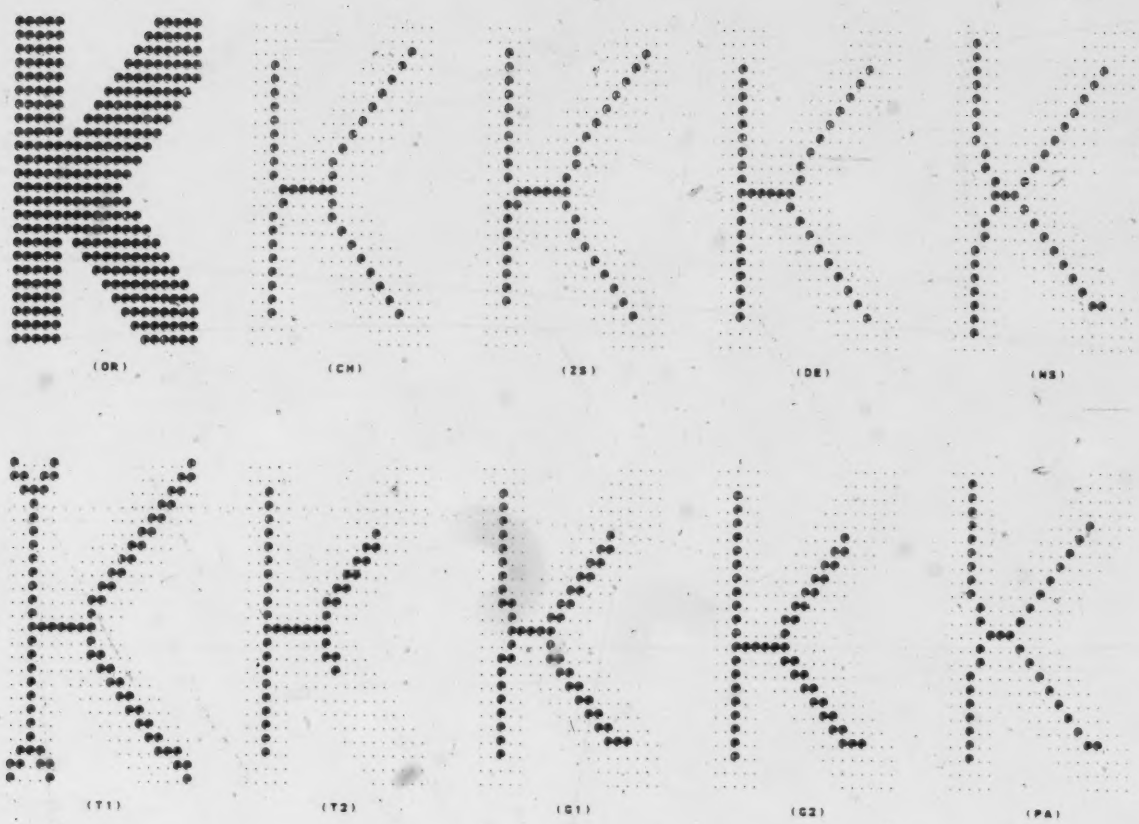

Figure A2.11 Skeletons of 'K'

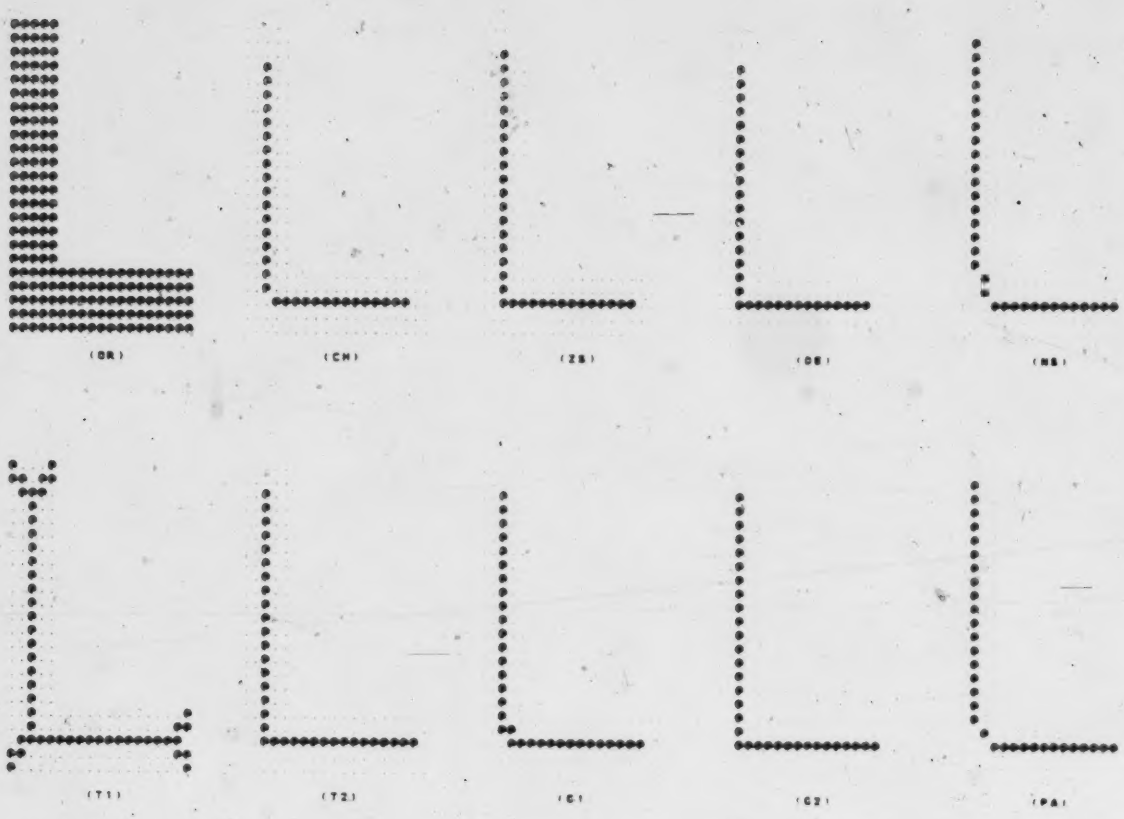

Figure A2.12 Skeletons of 'L'

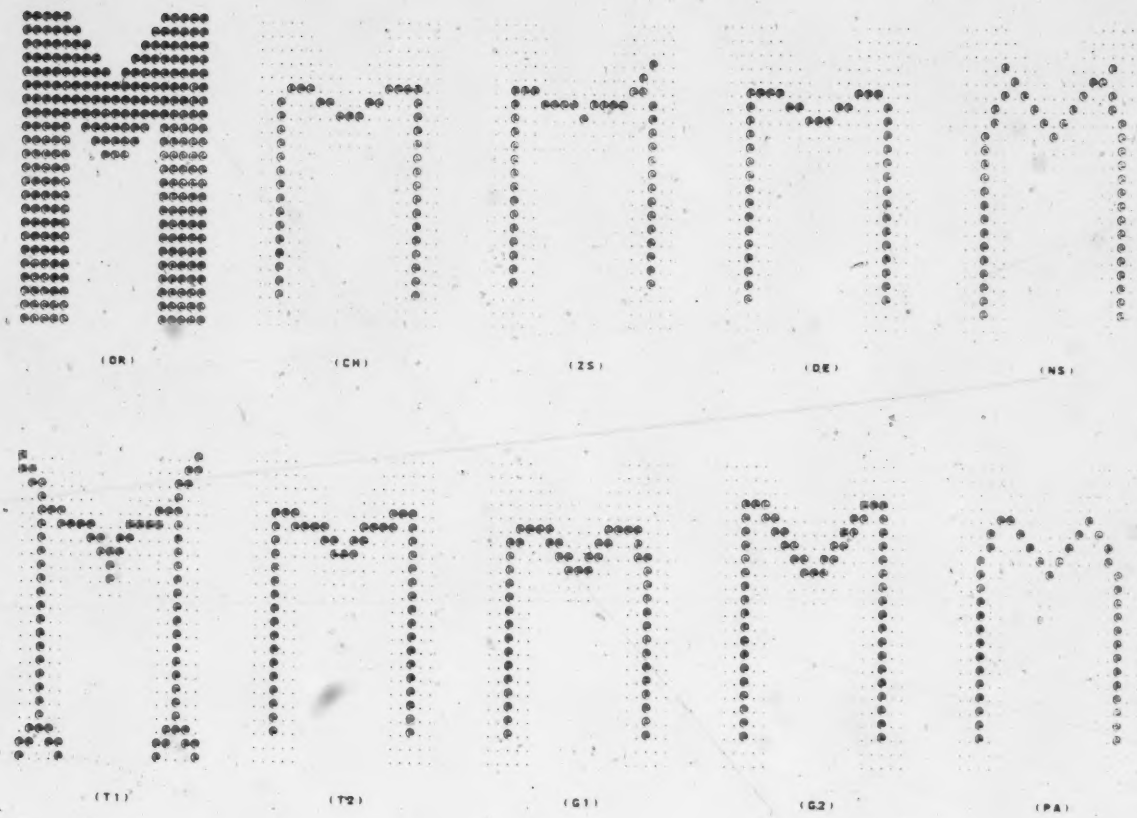

Figure A2.13 Skeletons of 'M'

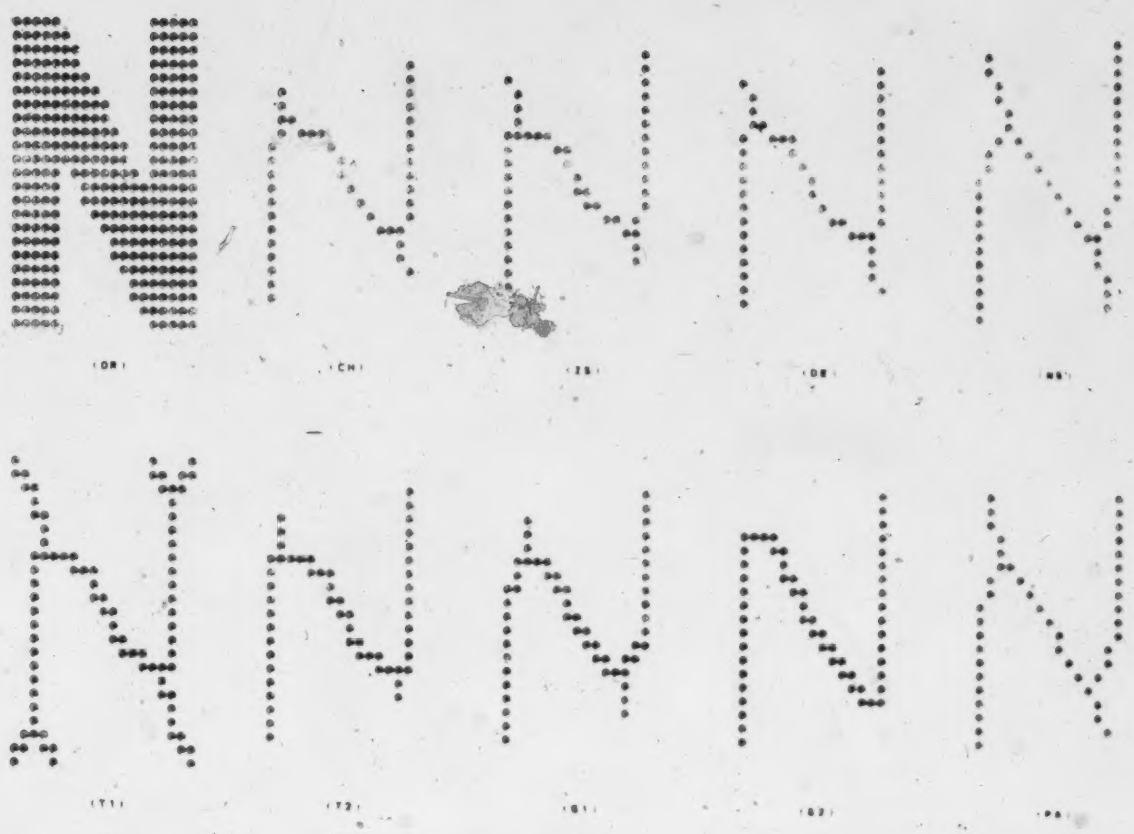

Figure A2.14 Skeletons of 'N'

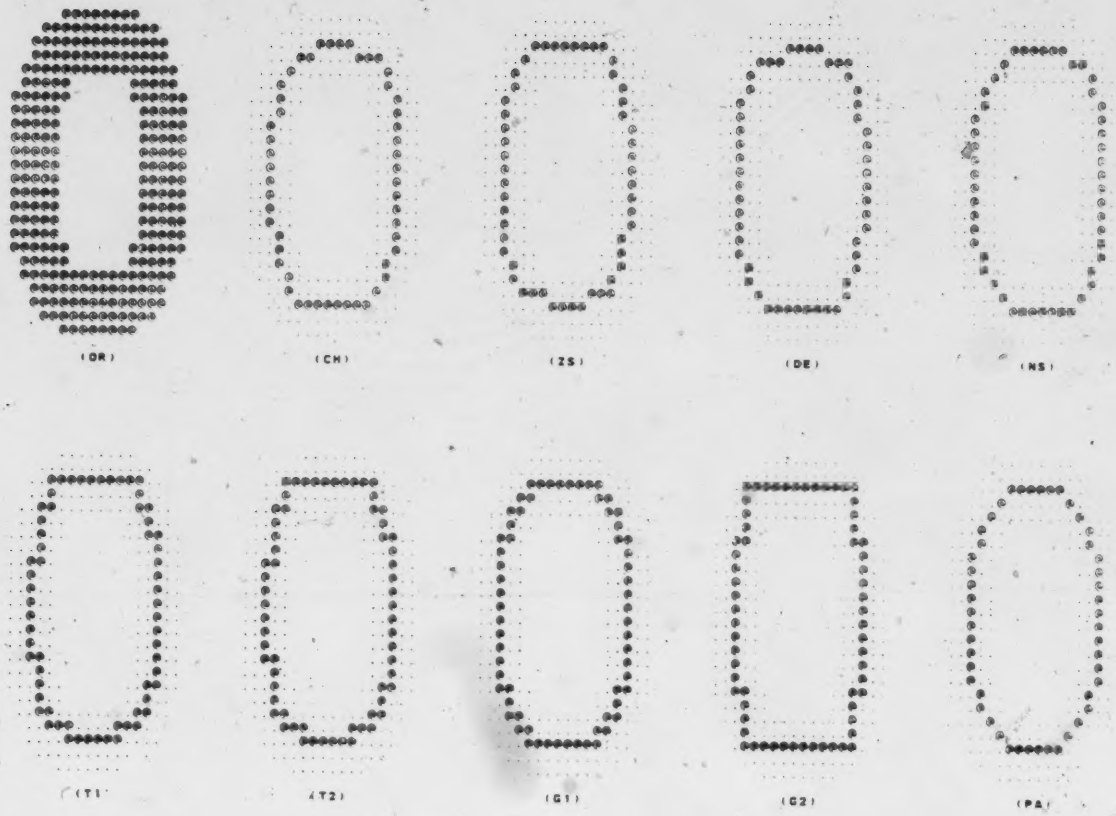

Figure A2.15 Skeletons of 'O'

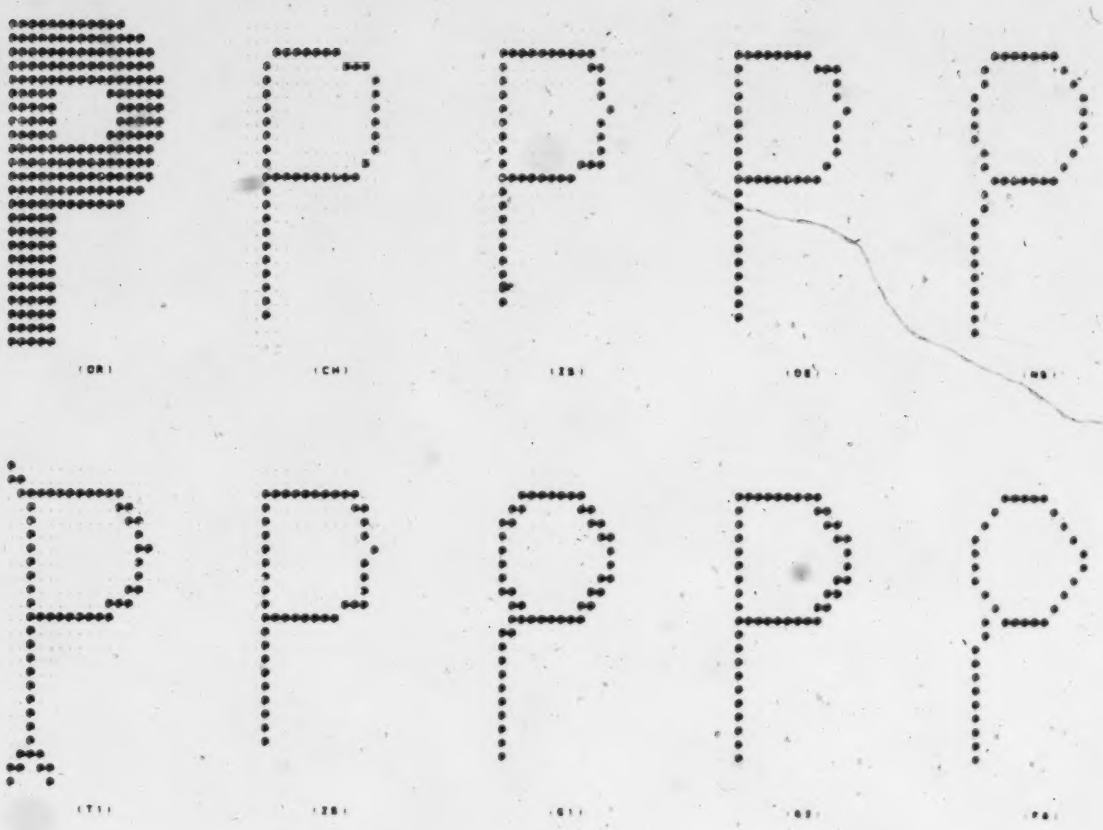

Figure A2.16 .Skeletons of 'P'

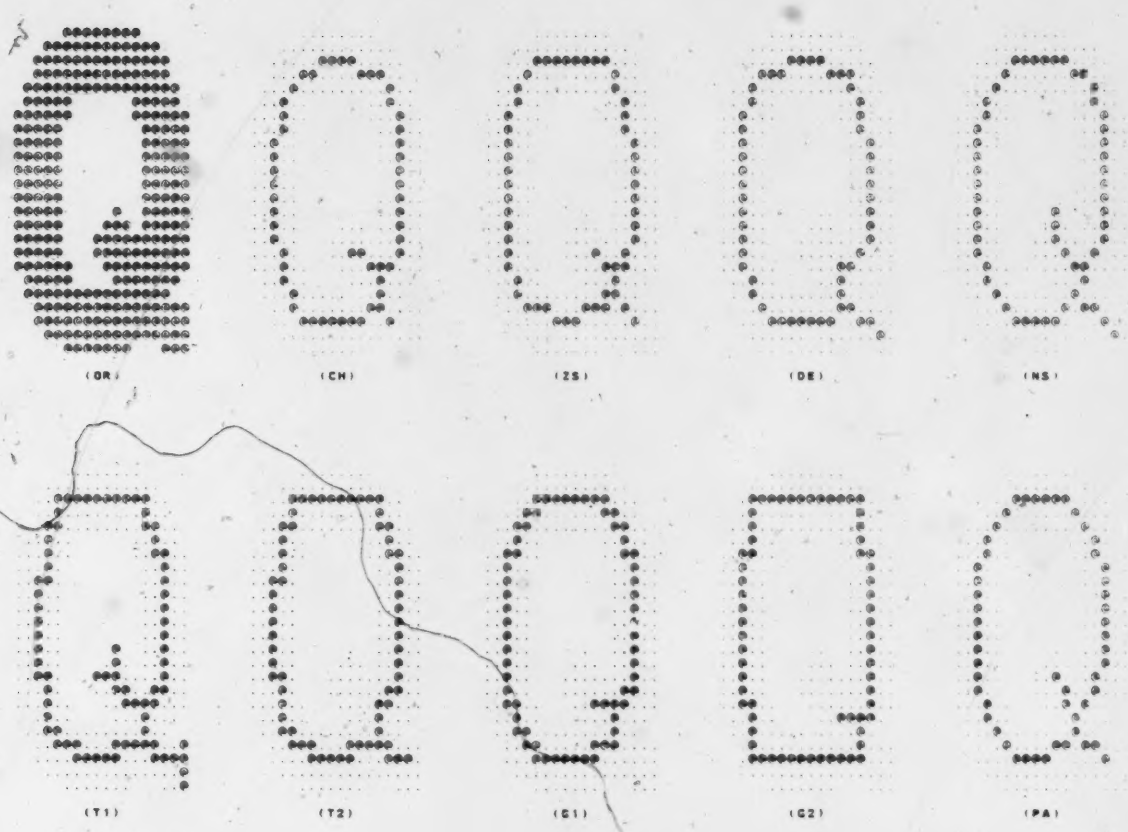

Figure A2.17 Skeletons of 'Q'

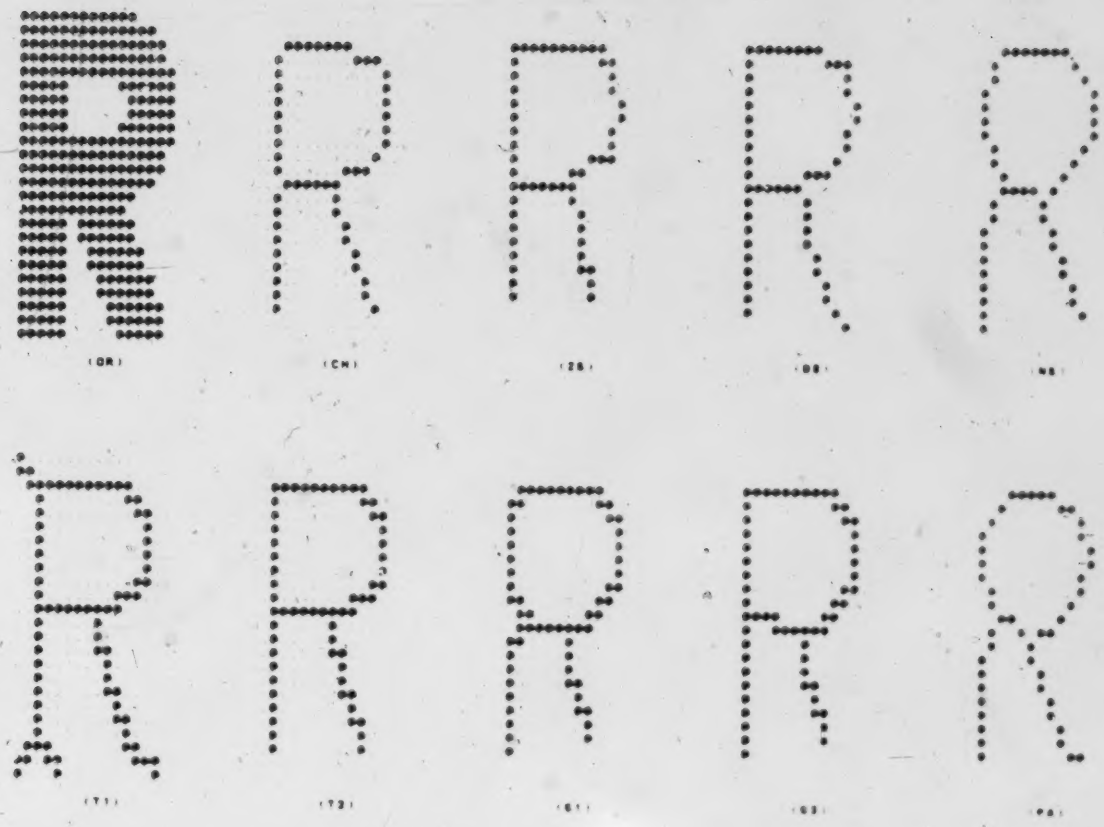

Figure A2.18 Skeletons of 'R'

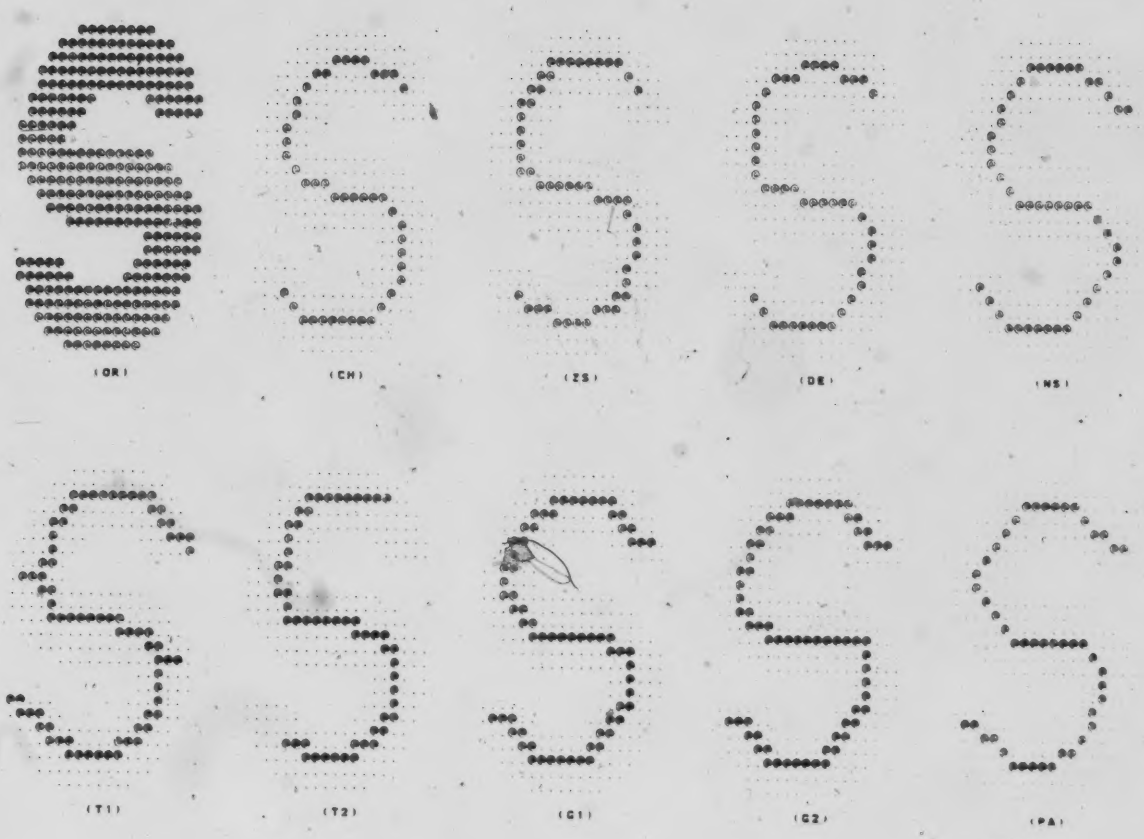

Figure A2.19 Skeletons of 'S'

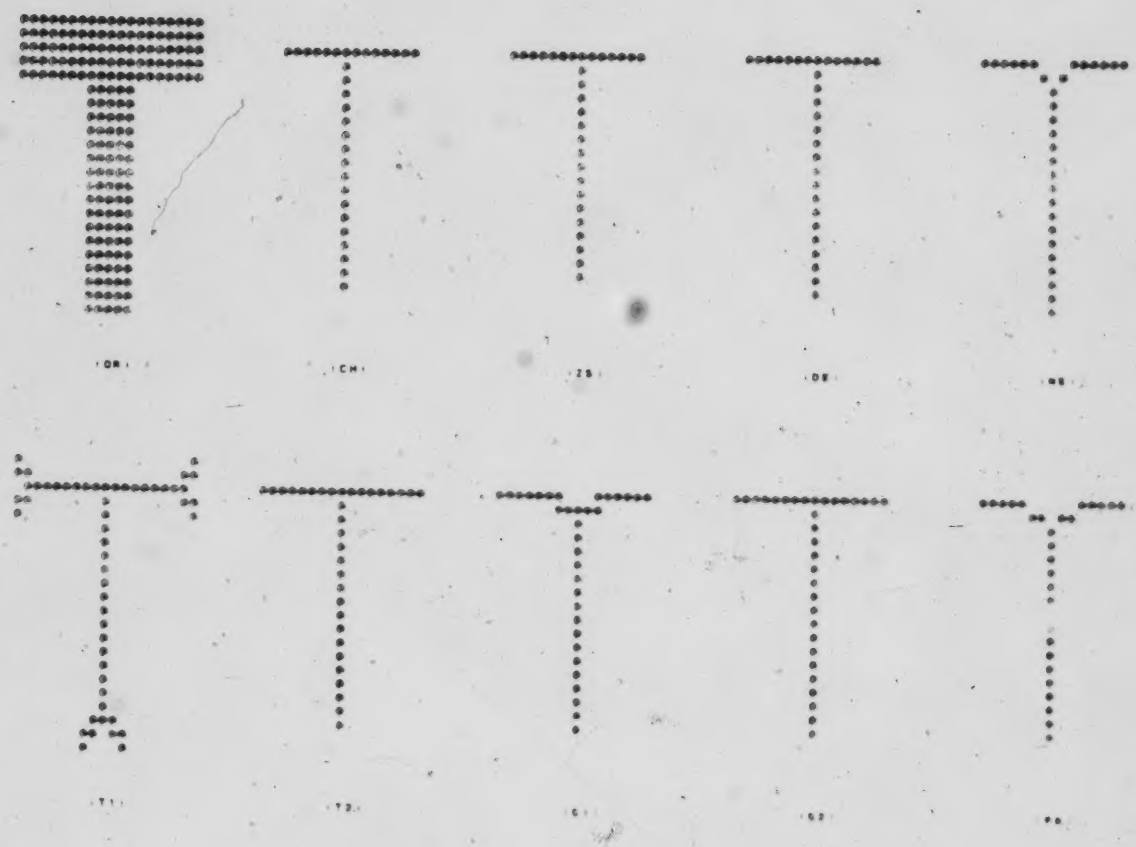

Figure A2.20 Skeletons of 'T'

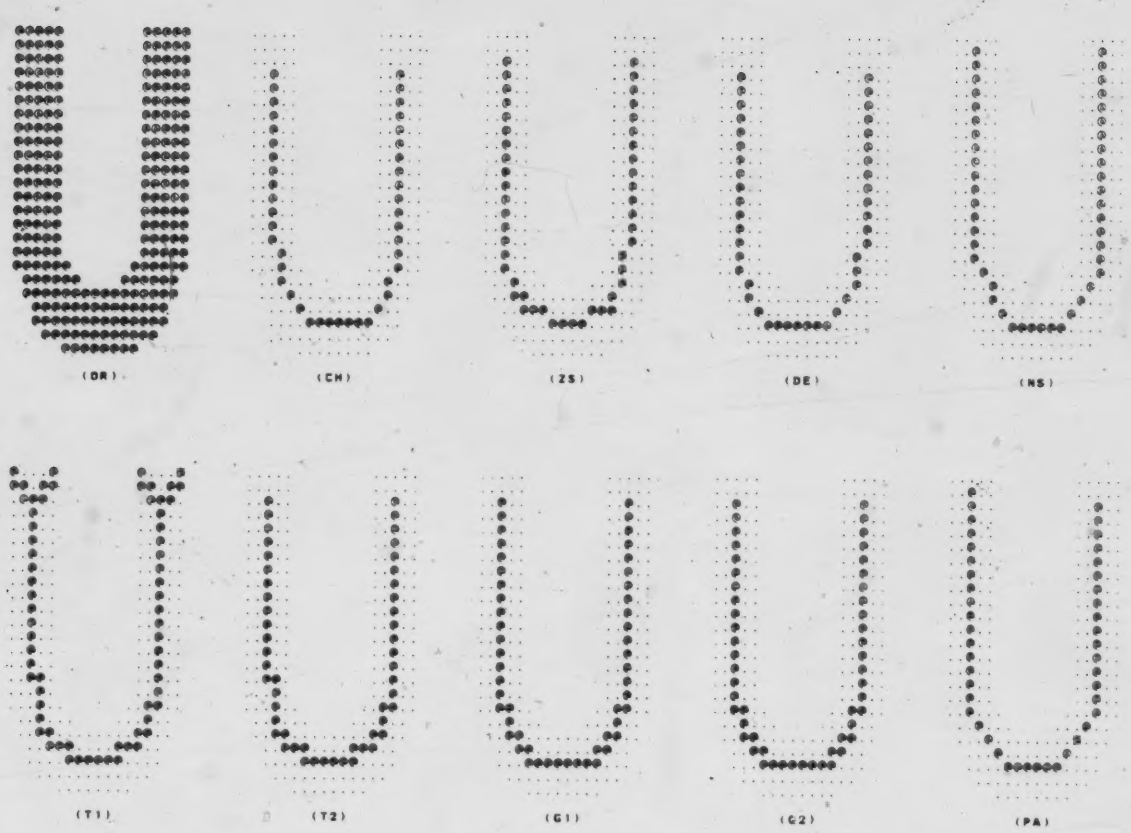

Figure A2.21 Skeletons of 'U'

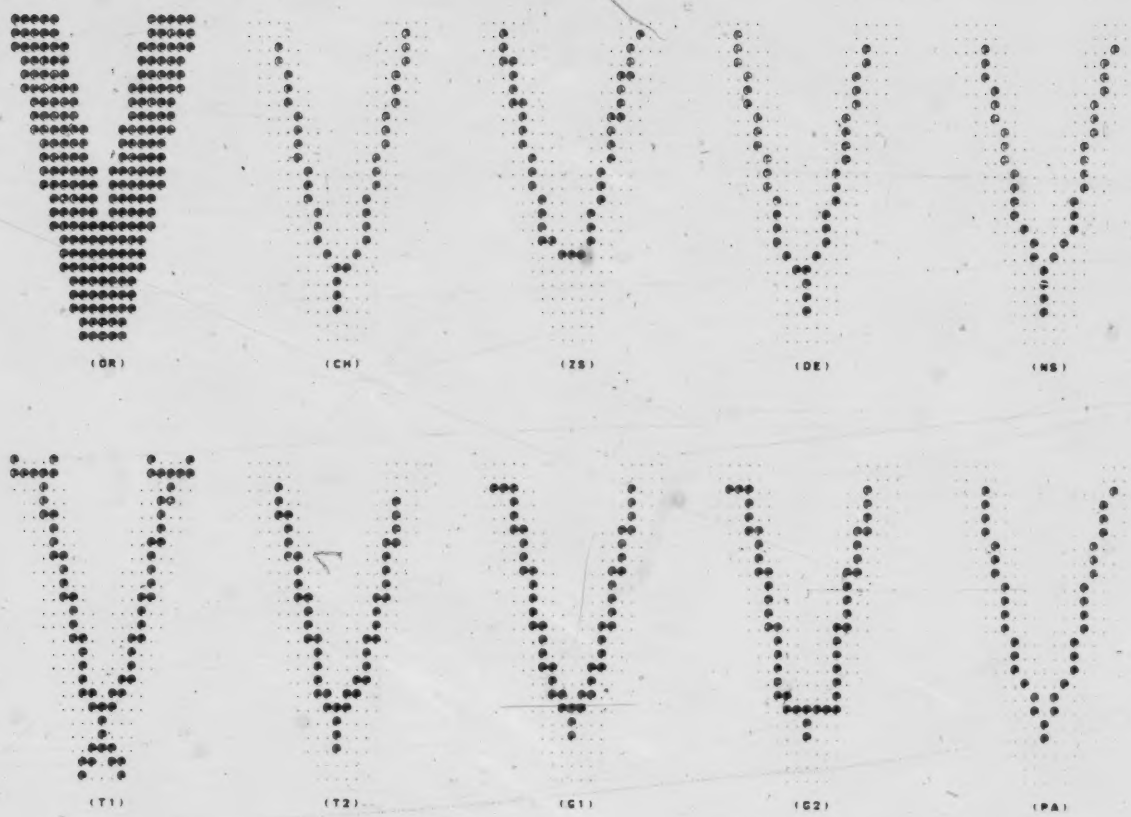

Figure A2.22 Skeletons of 'V'

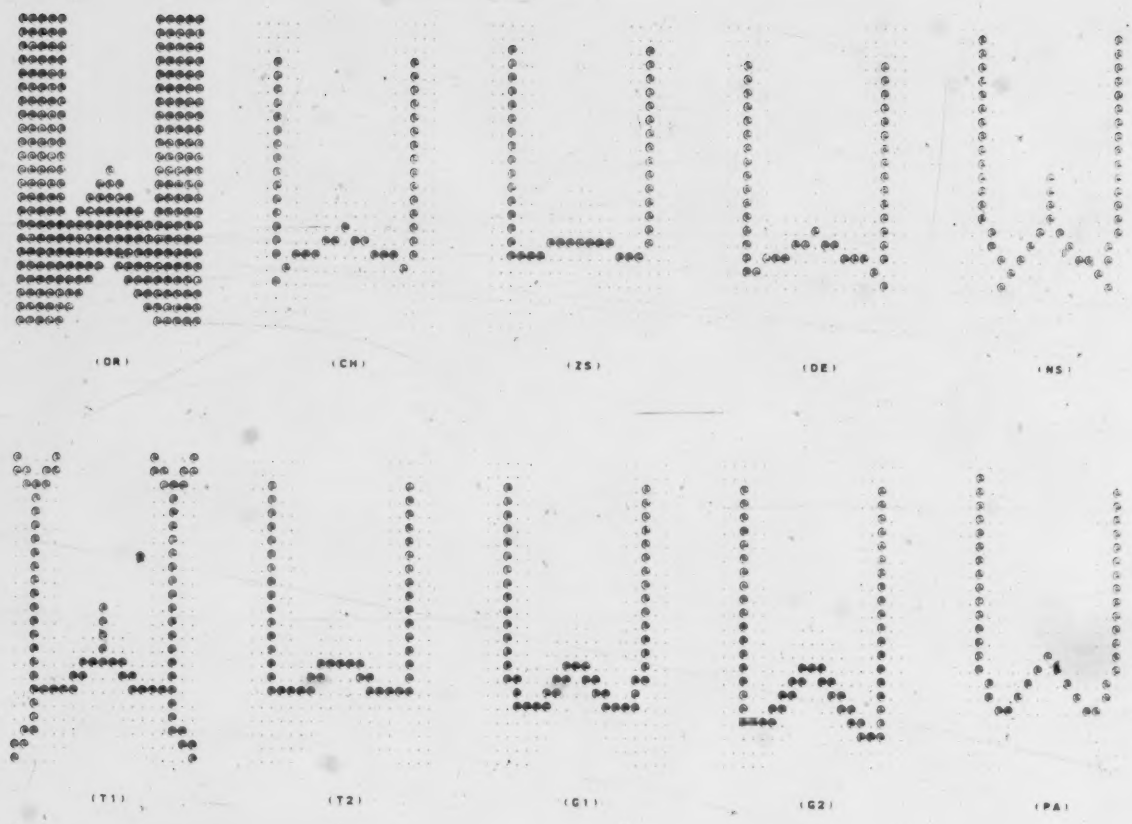

Figure A2.23 Skeletons of 'W'

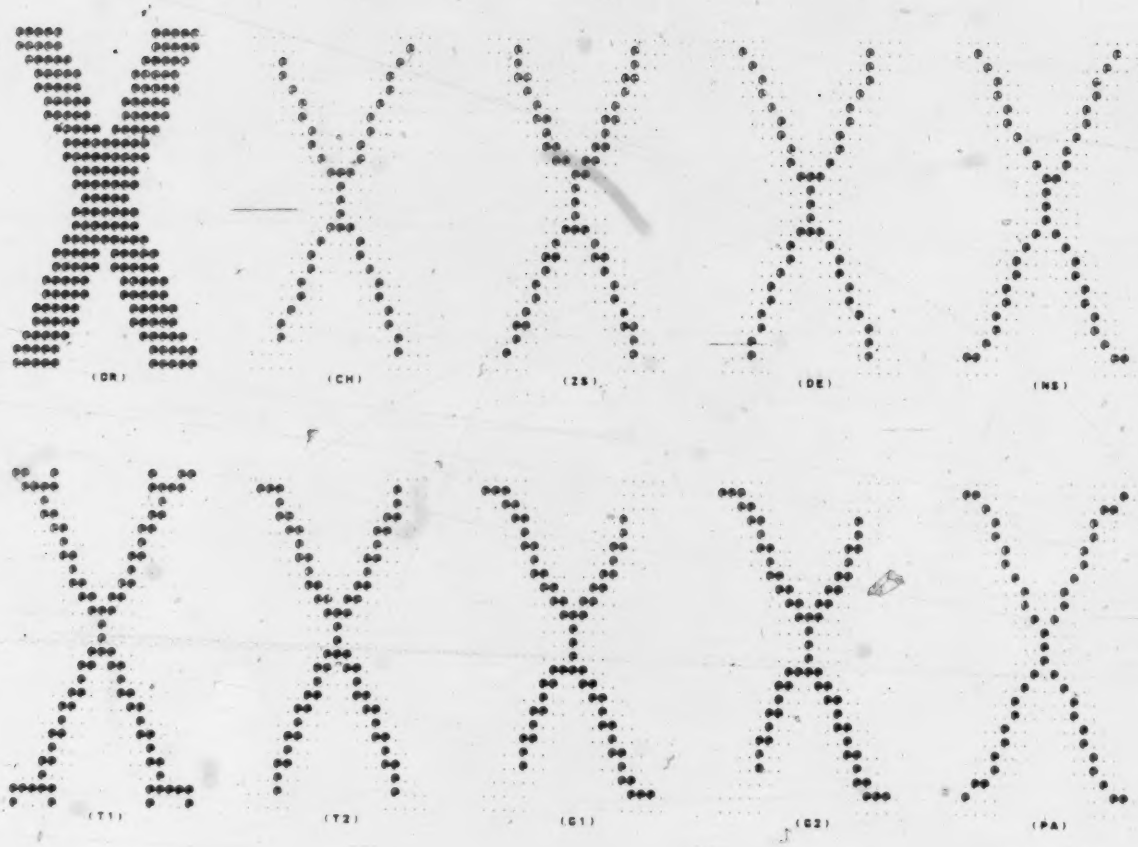

Figure A2.24 Skeletons of 'X'

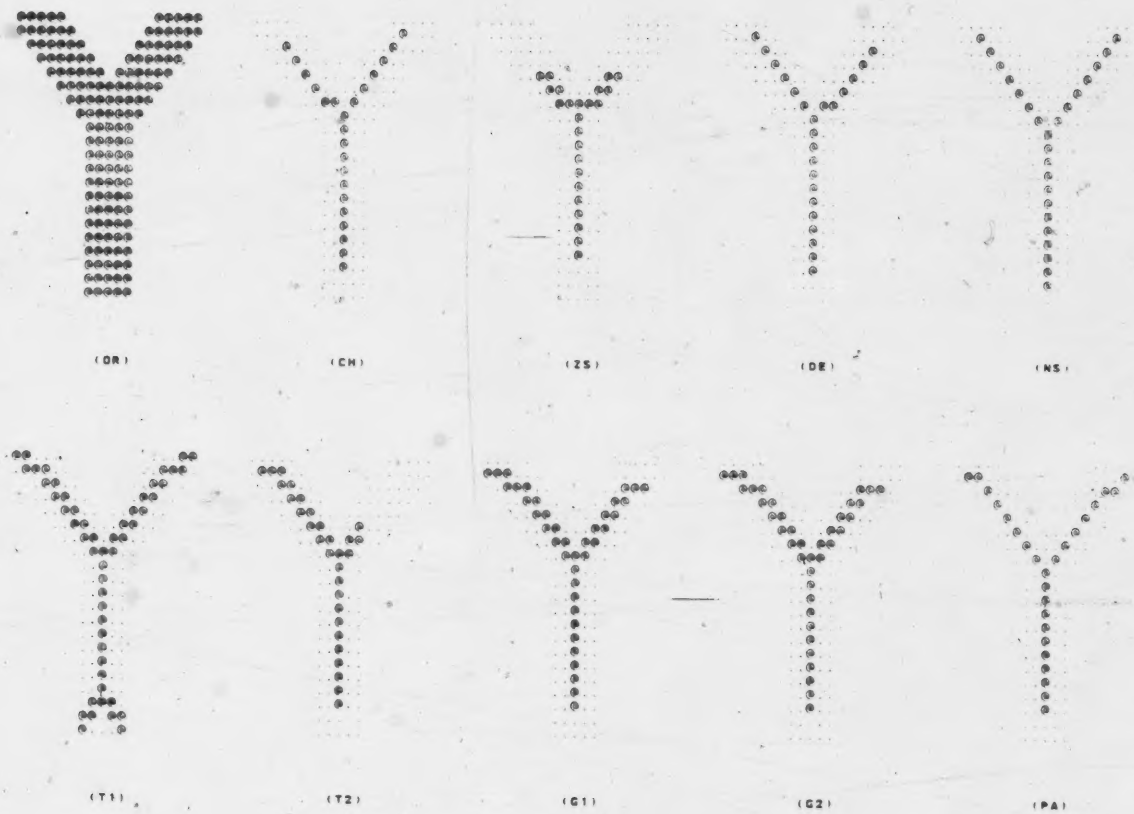

Figure A2.25 Skeletons of 'Y'

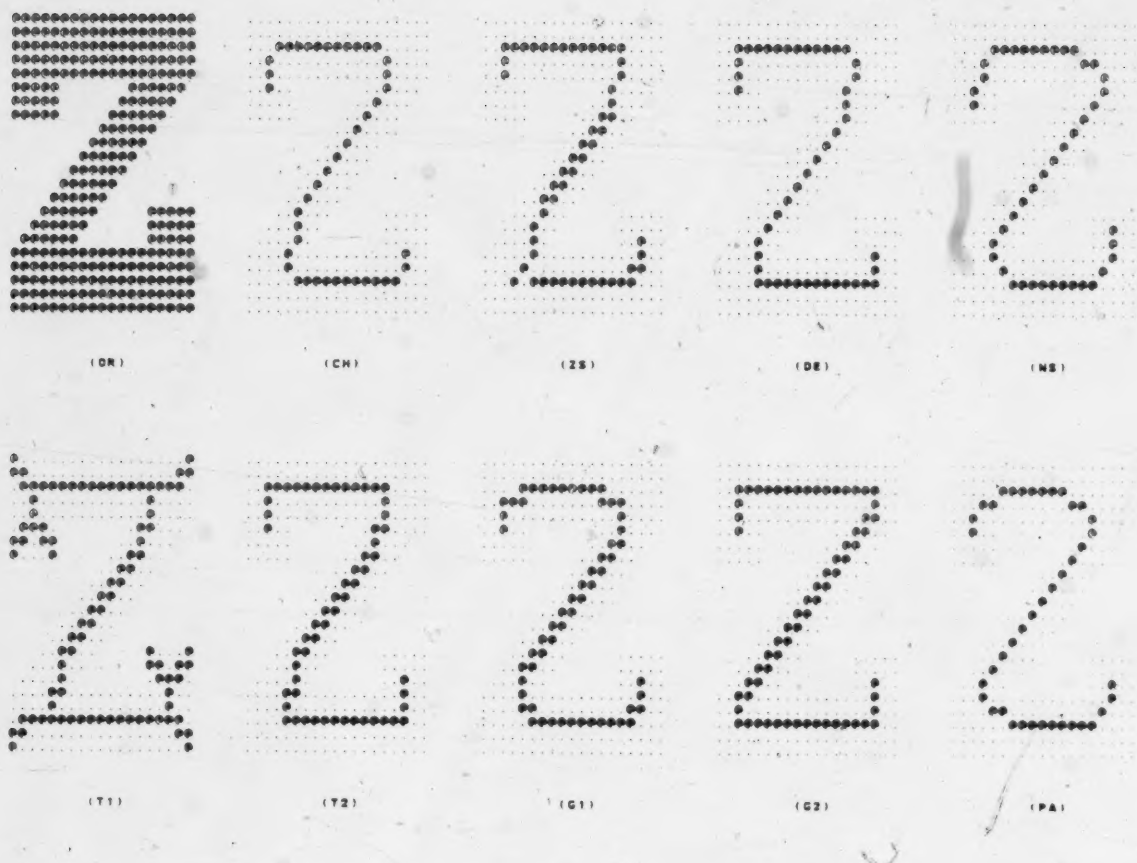

Figure A2.26 Skeletons of 'Z'

Appendix A3

Results of Experiment on Sensitivity

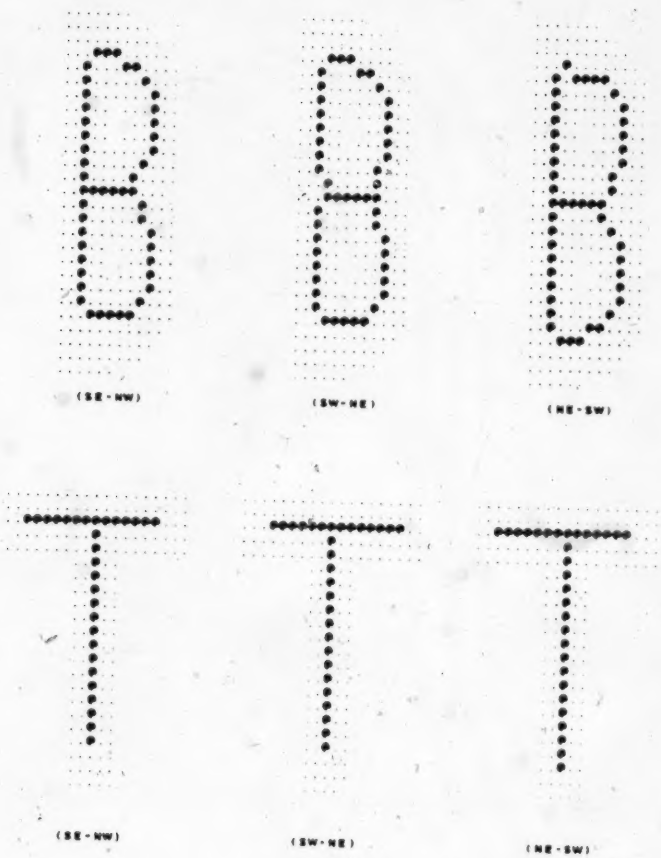

Figure A3.1 Skeletons produced by CH

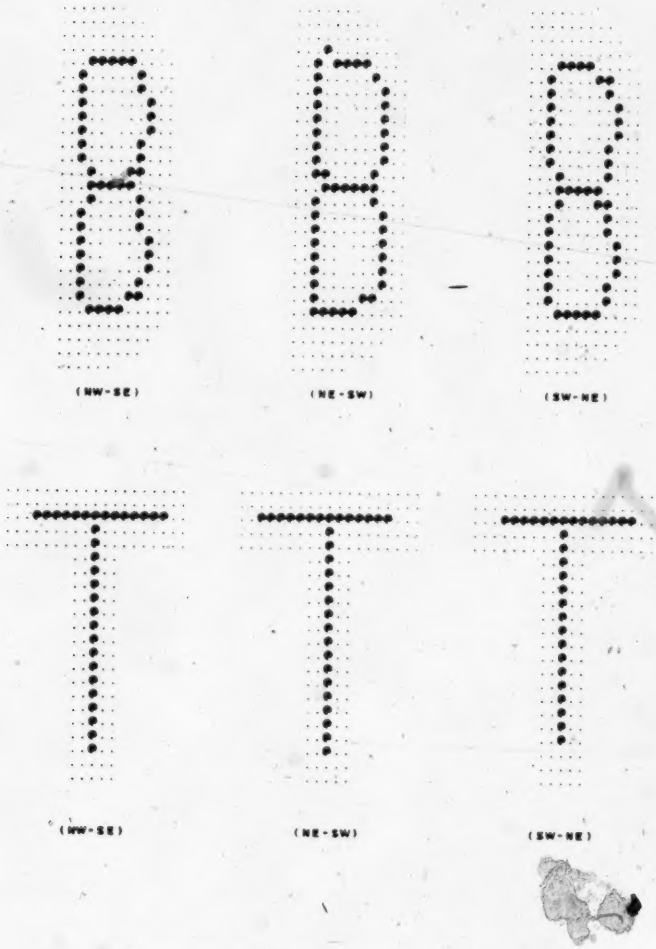

Figure A3.2 Skeletons produced by ZS

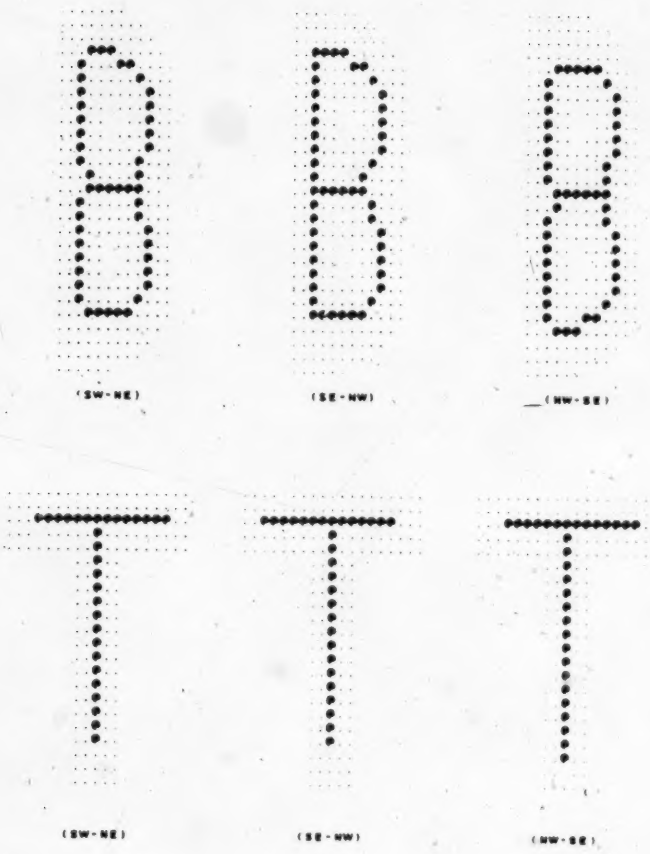

Figure A3.3 Skeletons produced by DE

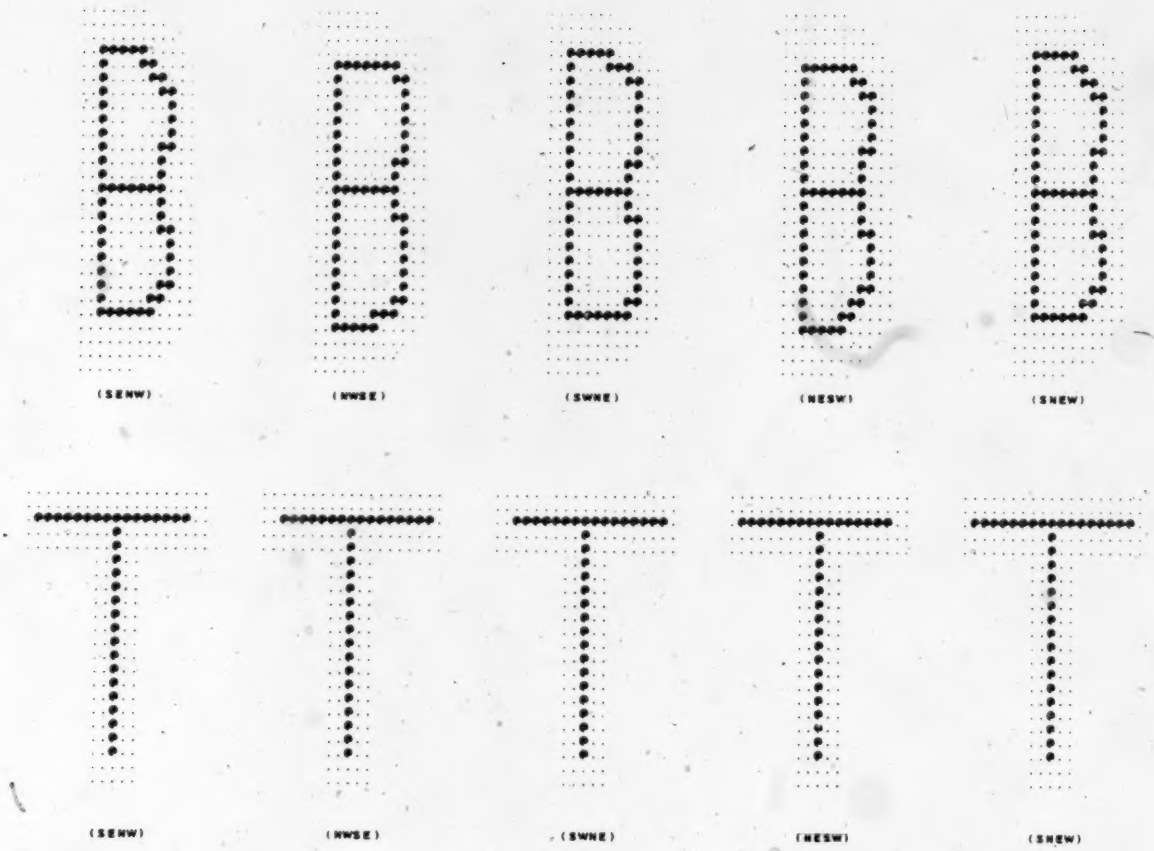

Figure A3.4 Skeletons produced by T2

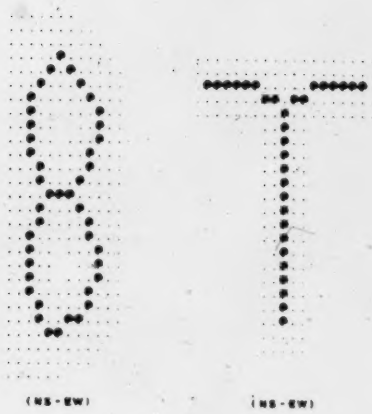

Figure A3.5 Skeletons produced by NS

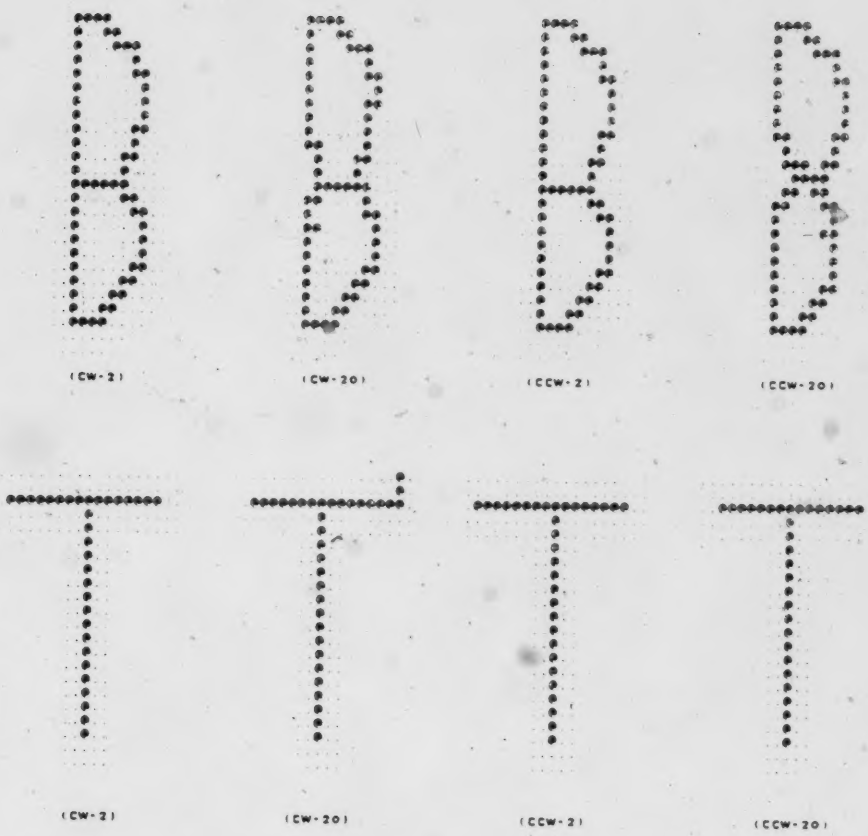

Figure A3.6 Skeletons produced by G2

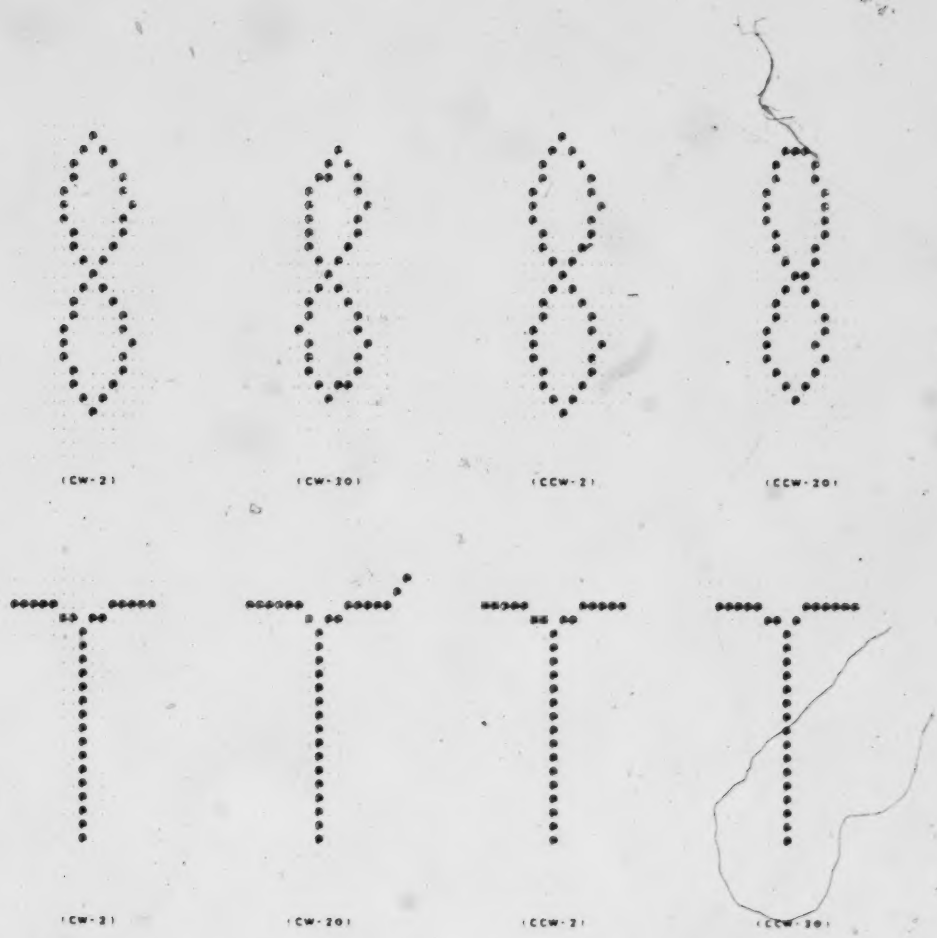

Figure A3.7 Skeletons produced by PA

Appendix A4

Results of Reformed CH, T2 and G2

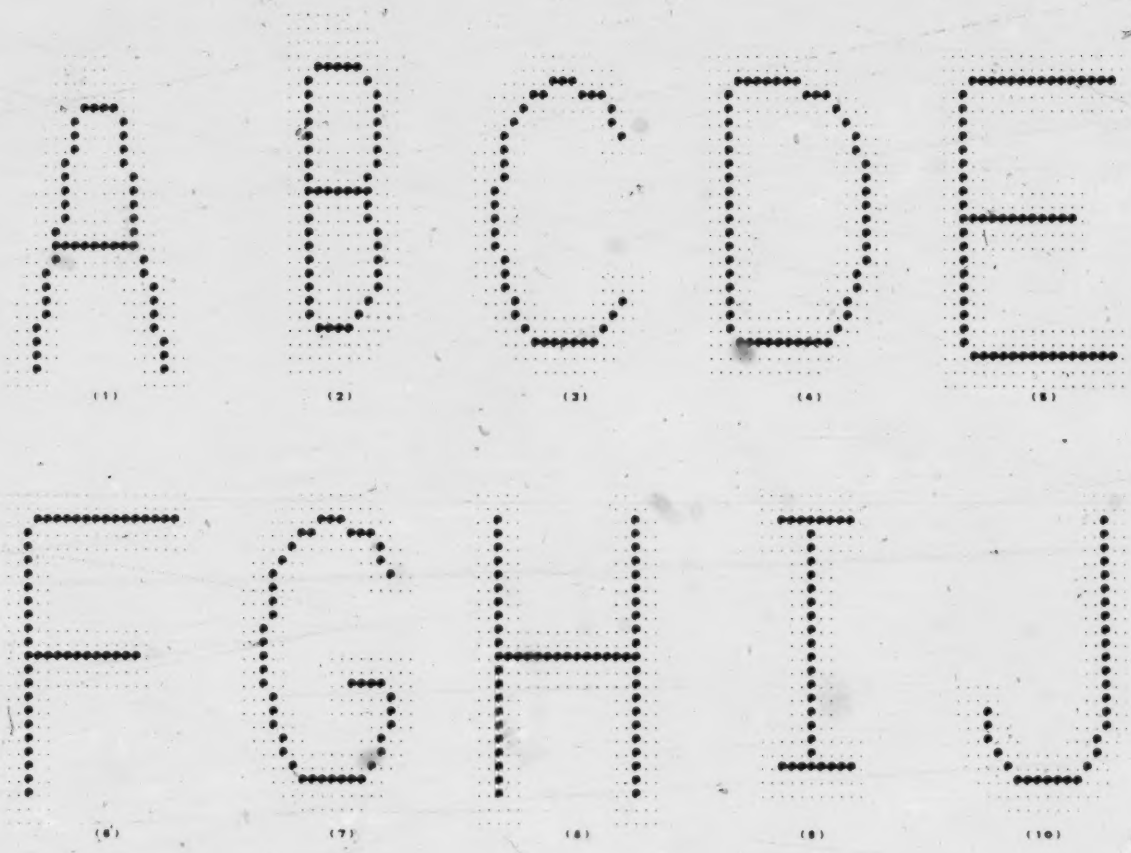

Figure A4.1 Skeletons produced by Reformed CH

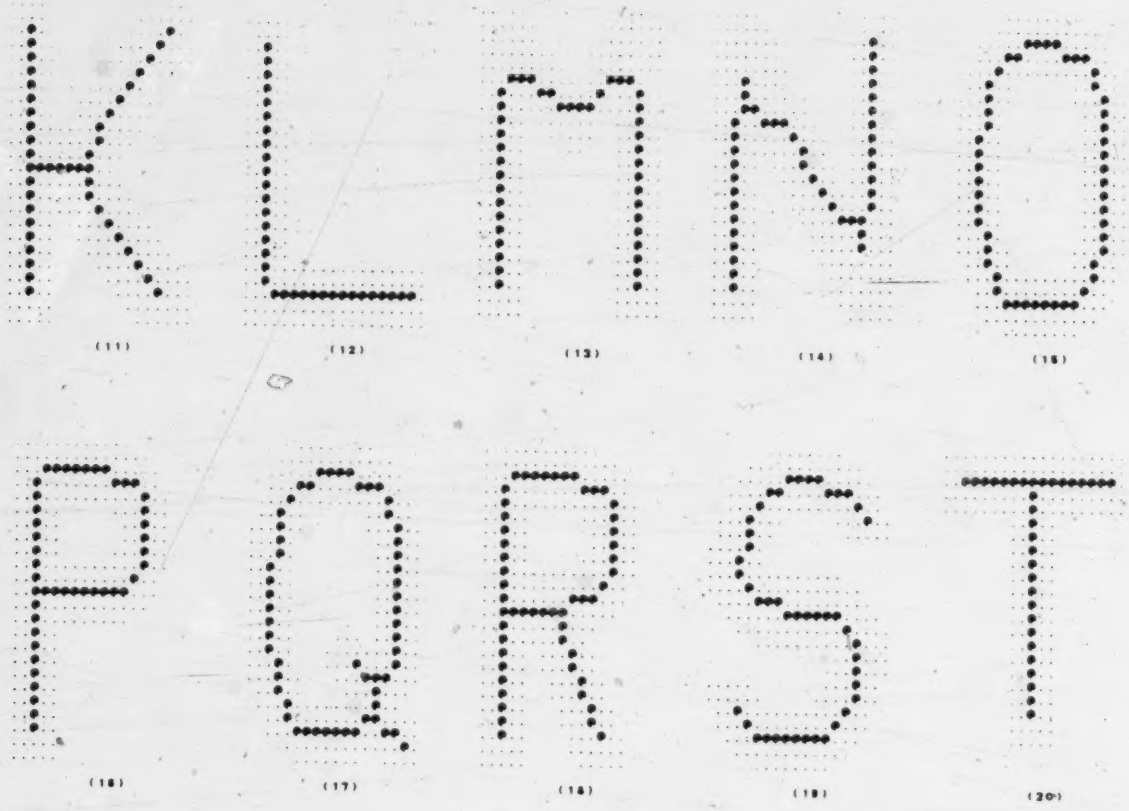

Figure A4.1. (cont.)

Figure A4.1. (cont.)

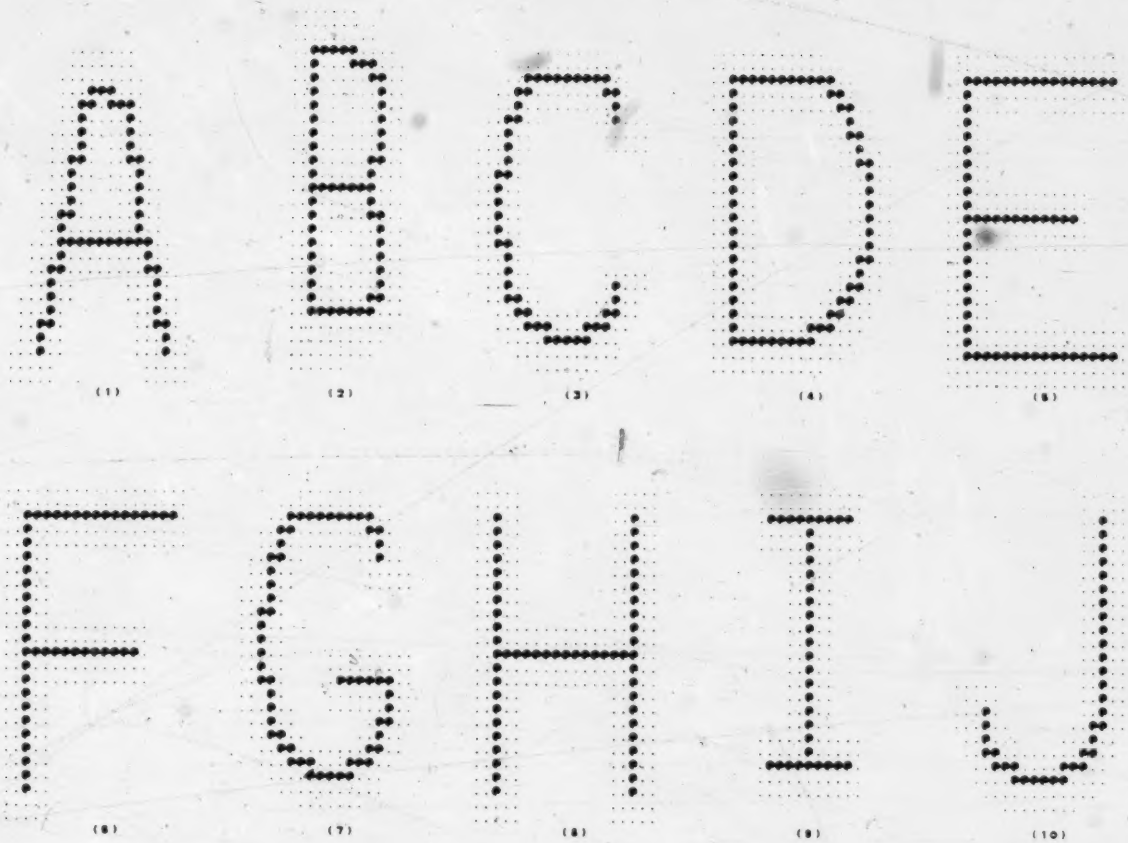

Figure A4.2 Skeletons produced by Reformed T2

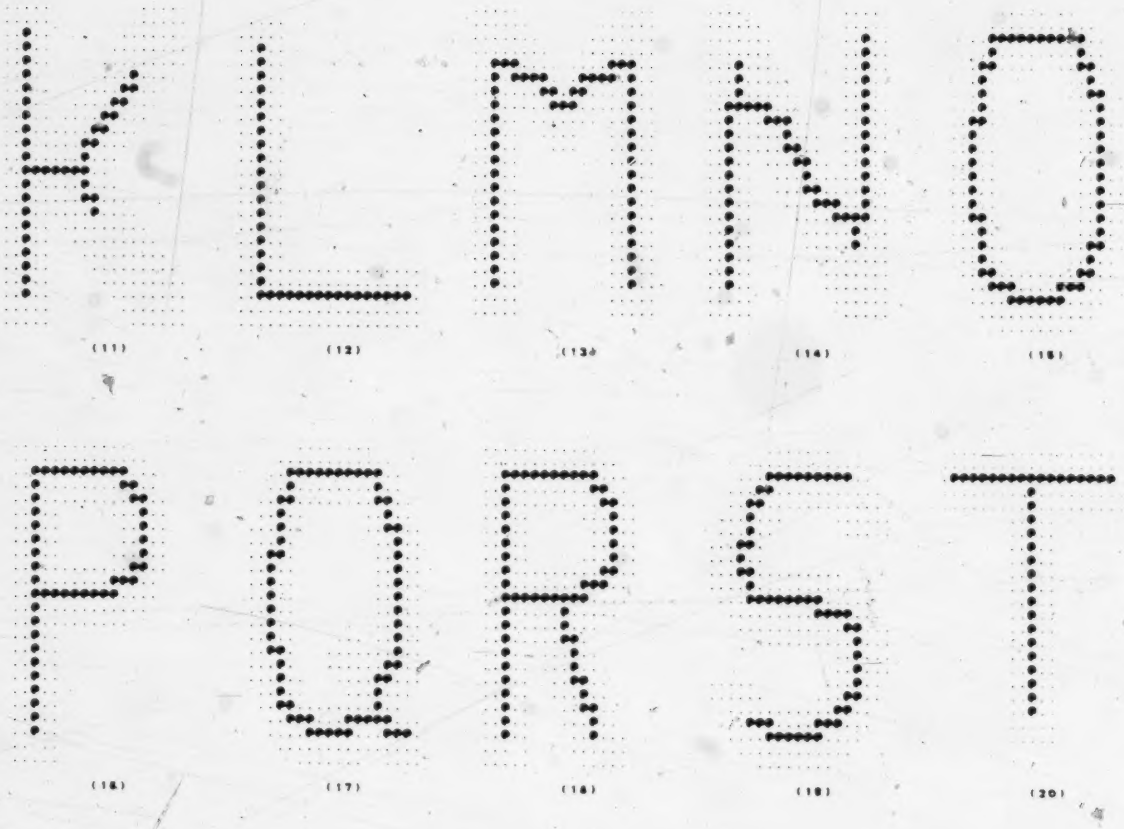

Figure A4.2. (cont.)

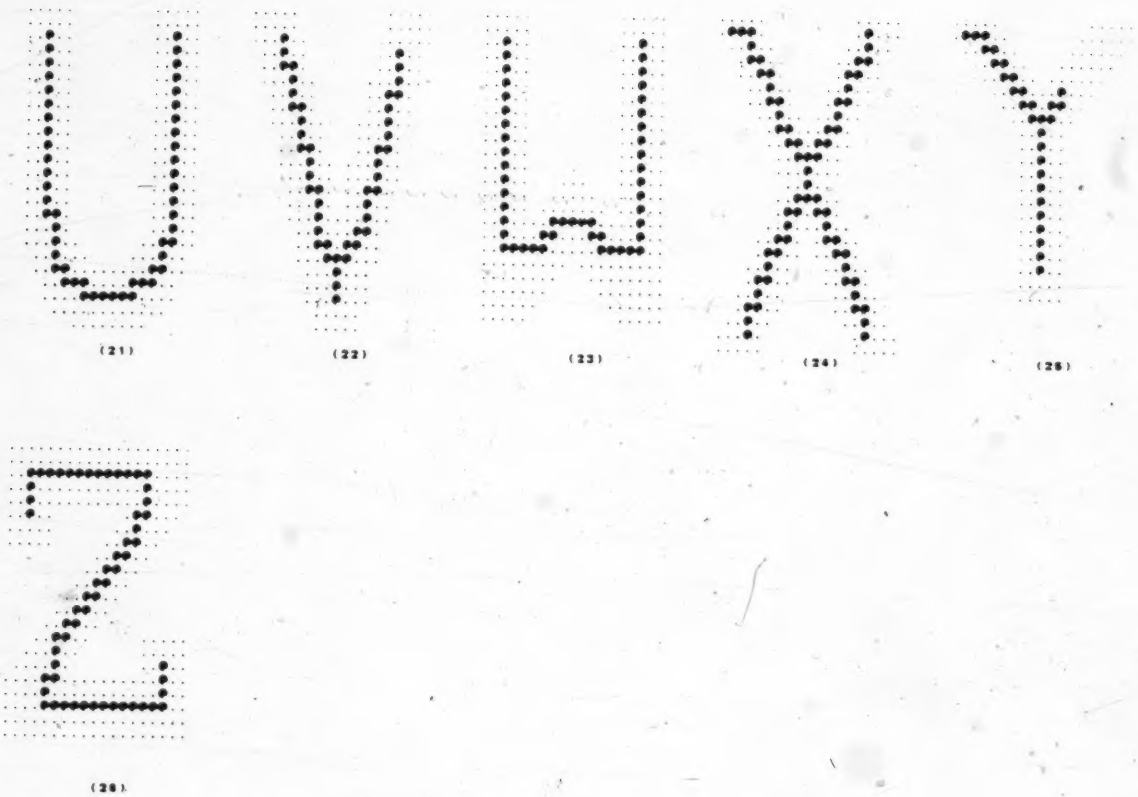

Figure A4.2. (cont.)

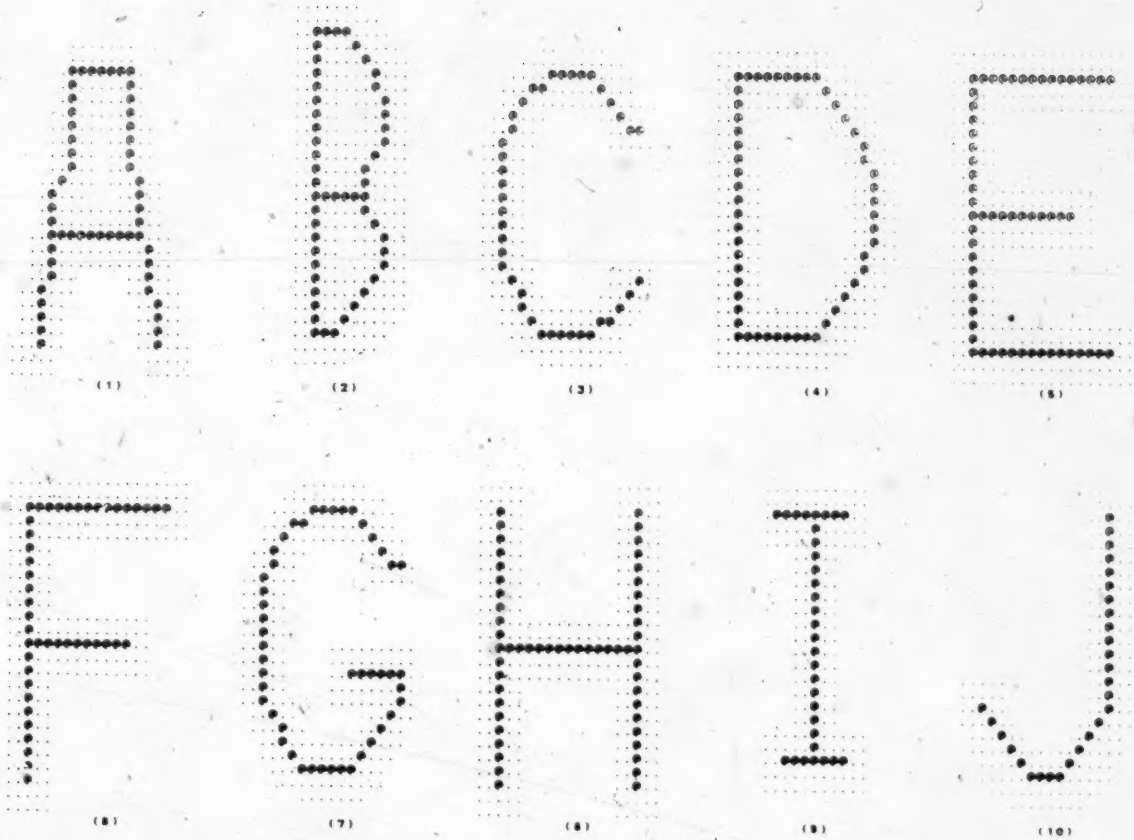

Figure A4.3 Skeletons produced by Reformed G2

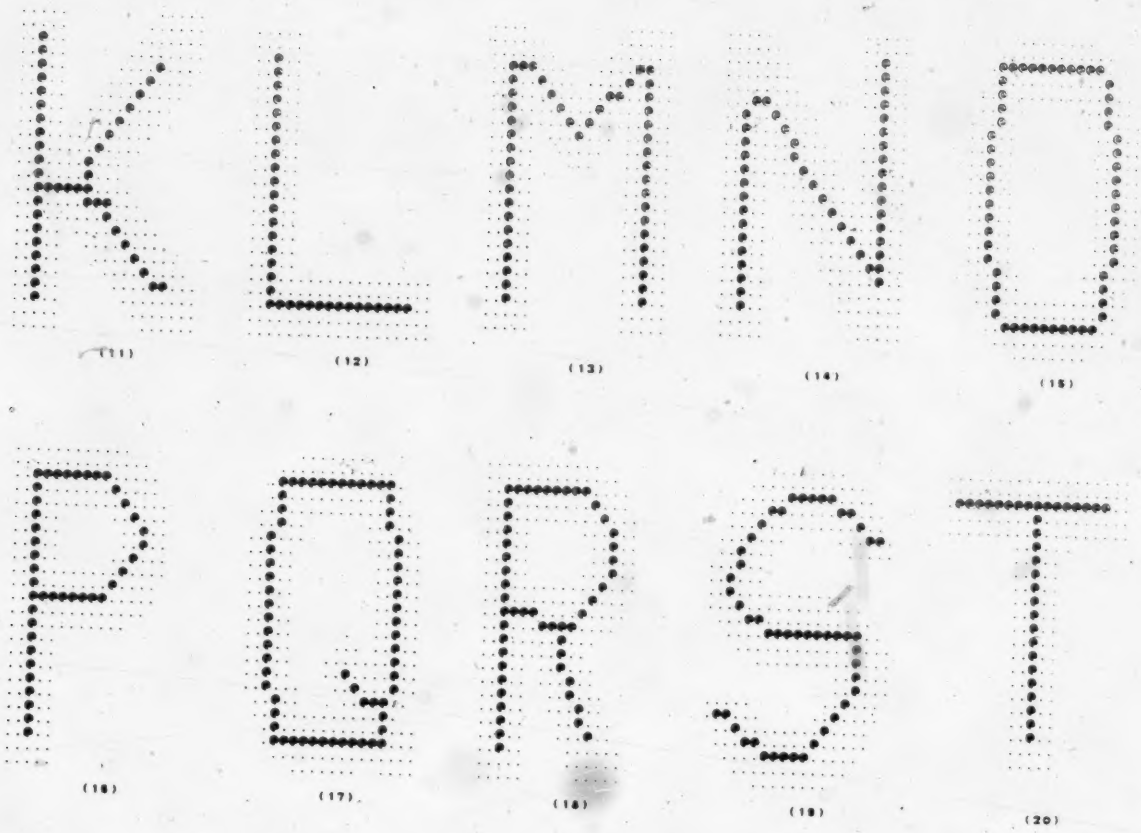

Figure A4.3 (cont.)

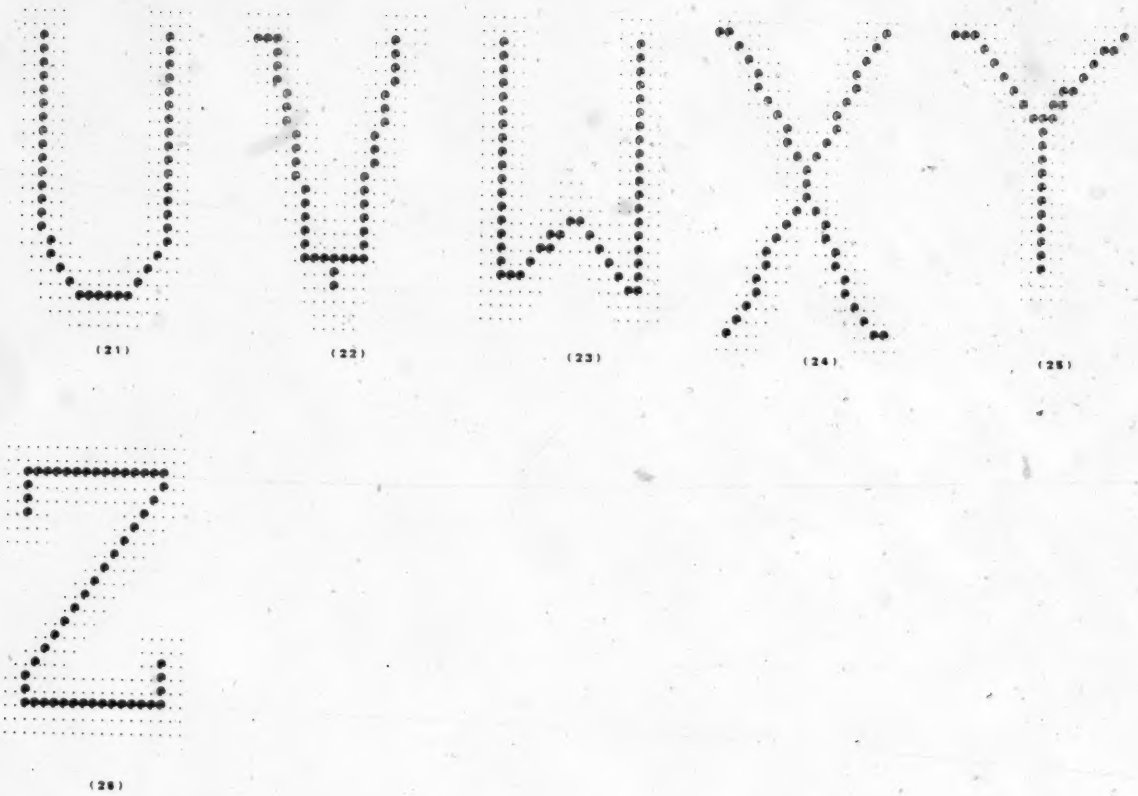

Figure A4.3. (cont.)